CKBS

AD-A240 805

(2)

DRAFT PROCEEDINGS

(VOLUME - 1)

DTIC
ELECTE
SEP 26 1991
S D

International Working Conference
on
COOPERATING KNOWLEDGE
BASED SYSTEMS

October 3-5, 1990
University of Keele
England

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

91-11597

91 9 26 012

Programme Committee

Prof. S. M. Deen (University of Keele, Chairman)
Prof. M. Brodie (CISL, Mass., USA)
Dr. D. Corkill (University of Mass., USA)
Prof. G. Gardarin (INRIA, France)
Prof. L. Gasser (University of S. Calif., USA)
Dr. M.N. Huhns (M.C.C., Austin, USA)
Prof. M. Jarke (University of Passau, FRG)
Prof. L. Kerschberg (GMU, USA)
Prof. J. Kouloumdjian (INSA-LYON, France)
Mr. B. Lepape (EEC ESPRIT Project)
Prof. V. R. Lesser (University of Mass., USA)
Prof. E. Mamdani (Queen Mary College, UK)
Dr. R. Manthey (ECRC, Munich, FRG)
Prof. J. Mylopoulos (Toronto University, Canada)
Prof. E. Neuhold (GMD, FRG)
Prof. M. Singh (UMIST, UK)
Prof. Y. Vassiliou (ICS, Greece)
Dr. J. Widom (IBM Almaden, USA)
Dr. K. Yokota (ICOT, Tokyo, Japan)

Organising Committee

Prof. S. Misbah Deen - Chairman
Dr. O. Pearl Brereton - Secretary
Dr. Chris Hawksley - Treasurer
Mr. Alan Treherne
Mr. Jonathan Knight
Dr. Glan Thomas

Sponsors

Information Technology Division of the Department of Trade and Industry,

BCS Data Management and Expert Systems specialist groups,

US Air Force European Office of Aerospace Research and Development,

European Research Office of the US Army

ESPRIT

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By per form 50
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

CONTENTS (VOLUME - 1)

Distributed Truth Maintenance

Michael N. Huhns and David M. Bridgeland

Microelectronics and Computer Technology Corporation
Artificial Intelligence Laboratory
3500 West Balcones Center Drive
Austin, TX, U.S.A. 78759-6509
huhns@mcc.com

7 September 1990

Abstract

In this paper we define the concept of logical consistency of belief among a group of computational agents that are able to reason nonmonotonically. We then provide an algorithm for truth maintenance that guarantees local consistency for each agent and global consistency for data shared by the agents. Furthermore, we show the algorithm to be complete, in the sense that if a consistent state exists, the algorithm will either find it or report failure. The algorithm has been implemented in the *RAD* distributed expert system shell.

1 Introduction

Two trends have recently become apparent out of the widespread use of knowledge-based systems: 1) systems are being developed for larger and more complicated domains, and 2) there are attempts to use several small systems in concert when their application domains overlap. Both of these trends argue for knowledge-based systems to be developed in a distributed fashion, where modules are constructed to interact productively. The individual modules then are characteristic of intelligent agents. The interconnected agents can cooperate in solving problems, share expertise, work in parallel on common problems, be developed modularly, be fault tolerant through redundancy, represent multiple viewpoints and the knowledge of multiple experts, and be reusable. Additional motivations are presented in [Huhns 1987] and [Gasser and Huhns 1989]. But in order for these agents to coordinate their activities and cooperate in solving mutual problems, it is essential that they be able to communicate with each other. Further, in order for them to interact intelligently and efficiently, we believe that the agents must be able to assess and maintain the integrity of the communicated information, as well as of their own knowledge.

2 Knowledge Base Integrity

There are many desirable properties for the knowledge base of an expert system or agent, such as completeness, conciseness, accuracy, and efficiency. For an agent that can reason nonmonotonically, there are additional properties used to describe the *integrity* of the agent's knowledge base: stability, well-foundedness, and logical consistency. A *stable* state of a knowledge base is one in which 1) each knowledge base element that has a valid justification is believed, and 2) each knowledge base element that lacks a valid justification is disbelieved. A *well-founded* knowledge base permits no set of its beliefs to be mutually dependent. A *logically-consistent* knowledge base is one that is stable at the time that consistency is determined and in which no logical contradiction exists. Depending on how beliefs, justifications, and data are represented, a consistent knowledge base may be one in which no datum is both believed and disbelieved (or neither), or in which no datum and its negation are both believed. These concepts are often extended to include

other types of contradictions

In addition, any algor thm that attempts to maintain well-founded stable states of a knowledge base, such as one of the many algorithms for truth maintenance [Doyle 1979, de Kleer 1986, Martins and Shapiro 1988, McAllester 1980, Russino T 1985], should be *complete*, in the sense that if a well-founded stable state exists, the algorithm will either find it or report failure. In general, we des re each agent in a multiagent environment to have a complete algorithm for maintaining the integrity of its own knowledge base.

However, the above definitions of proper ies for a single knowledge base are insufficient to characterize the multiple knowledge bases in such a multiagent environment. When agents that are nonmonotonic reasoners exchange beliefs and then make inferences based on the exchanged beliefs, then new concepts of knowledge-base integrity are needed. In addition, the relevant concept of global truth maintenance becomes especially problematic if agents must compute their beliefs locally, based on beliefs communicated and justified externally. The next sections extend the above definitions to the multiagent case.

2.1 The JTMS

We presume that each agent has a problem-solving component, separate from its knowledge base, that makes inferences and supplies the results to the knowledge base. Our discussion applies to the set of beliefs that are held and maintained in t'is knowledge base In particular, we focus on the systems for maintaining beliefs known as truth-maintenance systems (TMS) [Doyle 1979].

TMSs are a common way to achieve knowledge base integrity in a single agent system, because they deal with the frame problem, they deal with atomicity, and they lead to efficient search. Furthermore, the justification networks they create can be used for nonmonotonic reasoning, problem-solving explanations to a user, explanation-based learning, and multiagent negotiation. Our research is based on a justification-based TMS, in which every datum has a set of justifications and an associated status of IN (believed) or OUT (disbelieved).

In the example considered below, an initial state of a distributed knowledge base is given and presumed consistent Our goal is to construct a consistent extension of this state or determine that no such extension exists

The distributed TMS (DTMS) algorithm presented for this task is most often invoked to restore consistency when a consistent state is disrupted by altering the justification for a datum.

2.2 Consistent Beliefs among Agents

Consider a network of many agents, each with a partially-independent system of beliefs. The agents interact by exchanging data, either unsolicited or in response to a query. Each agent has two kinds of data in its knowledge base

Shared Data Beliefs that the agent has shared with another agent sometime in the past.

Private Data Beliefs that the agent has never shared with another agent

A private datum might become a shared datum by being told to another agent, or by being the answer to some other agent's query Once shared with other agents, a datum can never again be private. Each shared datum is shared by a subset of the agents in the network—precisely those that have either sent or received assertions about the datum.

We extend the concept of knowledge-base consistency stated above by defining four degrees of consistency and well-foundedness that are possible in a multiagent system

Inconsistency: one or more agents are individually inconsistent, i e , at least one agent has a private datum without a valid justification and labeled IN, or a private datum with a valid justification and labeled OUT.

Local Consistency: each agent is locally consistent, i e., no private OUT datum has a valid justification, and each private IN datum has a valid justification. However, there may be global inconsistency among agents. there may be a shared datum that one agent believes to be IN and another believes to be OUT.

Local-and-Shared Consistency: each agent is locally consistent and groups of agents are mutually consistent about any data they all share, i.e , each shared datum is either IN in all the agents that share it or OUT in those agents. There is, however, no global consistency.

Global Consistency: the agents are both individually and mutually consistent, i.e., their beliefs could be merged into one large knowledge base without the status of any datum necessarily changing.

In the absence of interagent communication, and presuming the local environment of each agent is consistent, then Local Consistency should hold. The introduction of interagent communication, however, tends to drive the system towards Inconsistency, because the agents might receive data that conflict with their current beliefs. The mechanism for truth maintenance we describe below enables each agent then to strive for Local-and-Shared Consistency. The presumption here is that the shared data are the most important, because they affect the problem solving of another agent, and so special effort should be made to maintain their consistency.

Although our goal is to maintain Local-and-Shared Consistency, we at times allow the system to fall short of this goal in order to permit agents to have different viewpoints. In this case, one agent may hold a belief that is known to be contrary to the belief of a second agent. The agents do not attempt to resolve this dispute if resolution would result in their being individually inconsistent. A consequence of this is that these agents should then not believe any data originating from each other, unless that agent can prove that its belief for that data is independent of the disputed data.

Ill-Foundedness: one or more agents have beliefs that are internally ill-founded

Local Well-Foundedness: each agent has beliefs that are internally well-founded; however, one or more agents may have shared data that are IN but have no valid justifications in any agent.

Local-and-Shared Well-Foundedness: each agent has beliefs that are internally well-founded, and every IN shared datum has a valid justification in some agent; however, there may be ill-founded circularities of beliefs among groups of agents.

Global Well-Foundedness: every datum has a globally valid justification and no set of data, whether local to an agent or distributed among a group of agents, is mutually dependent.

3 A Multiagent TMS

In the classical TMS, a datum can be either IN or OUT. For the DTMS, we refine the IN status to two new statuses: INTERNAL and EXTERNAL. An INTERNAL datum is one that is believed to be true, and that has a valid justification. An EXTERNAL datum is believed to be true, but need not have a valid justification. Intuitively, the justification of an EXTERNAL datum is "so-and-so told me." Hence, only a shared datum can be EXTERNAL. For Local-and-Shared Well-Foundedness, a shared datum must be INTERNAL to at least one of the agents that shares it and either INTERNAL or EXTERNAL to the rest of the agents.

The only justification-based TMS labeling algorithm known to be complete [Russinoff 1985] takes a generate and test approach, first unlabeling a collection of data, then attempting to relabel that collection. On failure to relabel, a superset of the last unlabeled collection is unlabeled. We take a similar approach in the DTMS. Since new data in one agent can change not only the status of that agent's beliefs, but also that of other agents, our unlabeling and subsequent labeling will sometimes involve multiple agents.

The support status of a shared datum is jointly maintained by several agents. Hence, a single agent is generally not free to change the status of a shared datum on its own accord. It must coordinate with the other agents so that they are all consistent on the status of the datum. Central to the DTMS then is the single agent operation of label-wrt. label-wrt is a variation of classical TMS labeling in which the statuses of some data—though unlabeled—are fixed by external requirements.

More precisely, label-wrt is given a network of data. Some of the data have statuses of IN, OUT, INTERNAL, or EXTERNAL. Other data are unlabeled. For each shared datum, there is a *desired* label of either OUT, INTERNAL, or EXTERNAL. label-wrt either finds a consistent well-founded labeling of the network that satisfies the shared data requirements, or it reports failure. Space prohibits a presentation of an algorithm to implement label-wrt. Our approach is a variation of Russinoff's well-founded and complete labeling algorithm [Russinoff 1985].

3.1 Algorithm Schema

The DTMS is a double generate and test Relabeling is invoked by the addition or removal of a justification. When invoked, the DTMS does the following three things:

1. Unlabel some data, including the newly justified datum and presumably its consequences. This unlabeled data set might be confined to a single agent or it might span several agents. If a shared datum is unlabeled in some agent, it must be unlabeled in all the agents that share it.
2. Choose labelings for all the unlabeled shared data, as defined above.
3. Label each of the affected agents with respect to the requirements imposed by the shared data, invoking label-wrt as described above. If any of the affected agents fails to label, then backtrack. Either choose different labelings for the shared data (step 2), or unlabel a different set of data (step 1)

This schema will be refined later, but some nice properties emerge at this abstract level:

- Any labeling found by the DTMS will have Local-and-Shared Consistency and Well-Foundedness.
- If the two generate steps are exhaustive, the DTMS is complete: it will find a labeling should one exist

Note that these properties are true both of the DTMS algorithm described in this paper, and any other algorithm that conforms to this schema

3.2 Unlabeling

When the DTMS algorithm is invoked, it starts by unlabeling a collection of data. This collection may be confined to a single agent or it may span many agents However, it must meet the following constraints:

1 It must include the datum that originally acquired the new justification

2 A shared datum that is unlabeled in one agent must be unlabeled in all the agents that share it

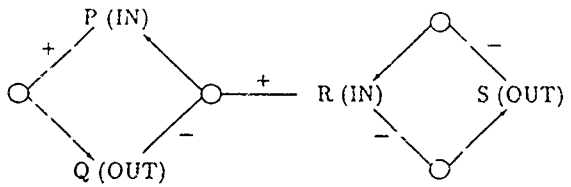


Figure 1: Relabeling upstream data to resolve an odd loop

3. On failure to label the collection, it must generate a new collection of unlabeled data. To guarantee completeness, the generation must be exhaustive: it must eventually generate a collection sufficiently large that failure to label it means the whole network cannot be labeled.

Using only these constraints, unlabeling is underconstrained: many algorithms satisfy. For example, on any status change one could unlabel all data in all agents. This global unlabeling satisfies all the constraints and is also quite simple, but also is too inefficient for practical use. The global unlabeling does reveal two DTMS principles that motivate the more complex algorithm presented later:

Principle 1 *Belief changes should be resolved with as few agents as possible.*

Principle 2 *Belief changes should be resolved by changing as few beliefs as possible.*

Most belief changes can be resolved by changing things only "downstream" of the new justification, i.e., those data that directly or indirectly depend on the datum newly justified. It is sometimes necessary to move "upstream" as well, and relabel data that directly or indirectly support the status of the newly justified datum. Consider the knowledge base for a single agent shown in Figure 1 [Russinoff 1985]. Here, datum Q acquires the new justification shown in dotted lines. If only P and Q are reassigned, the system is forced to report an unsatisfiable circularity. In order to restore stability, the status of the data upstream from P must be changed: if the system makes S `OUT` and R `IN`, both P and Q can be `OUT`.

Principle 3 *Belief changes should be resolved downstream if possible; upstream relabeling should be minimized.*

The above principles motivate the algorithm unlabel It attempts to minimize both the involvement of other agents and the unlabeling of upstream data, but prefers the former to the latter. Unlabel is invoked on a list containing either the newly justified datum, when unlabel is first invoked, or the unlabeled data that could not be labeled on a previous invocation. Unlabel attempts to find yet to be unlabeled private data downstream of those already labeled. If there are none, it looks for shared data downstream, unlabeling those in all the agents that share them, and also unlabeling private data downstream of the shared data. Finally, if there are no shared data downstream that are yet to be unlabeled, it unlabels data just upstream of all the downstream data, and all private data downstream of that. If there is nothing yet to be unlabeled upstream, unlabel fails and, in fact, the data admit no Local-and-Shared Consistent and Well-Founded labeling.

Consider the DTMS network in Figure 2. There are two agents, Agent 1 and Agent 2, and they share the datum T. As in Figure 1, the initial labeling shown in the diagram is perturbed by the addition of the new dotted justification. Agent 1 initially unlabels just the changed datum and private data downstream, P and Q, but there is no consistent relabeling. Hence, Agent 1 unlabels all shared data downstream of P and Q, and all private data downstream from there: P, Q, both Ts, and U. Again labeling fails. Since there is no further shared data downstream, Agent 1 and Agent 2 unlabel upstream and privately downstream from there: P, Q, Ts, U, R, and S. Now labeling succeeds (with S and U IN and everything else OUT). Had labeling failed, Unlabel would not be able to unlabel more data, and would report that the network is inconsistent.

3.2.1 Distributed System Issues

To be implemented, the unlabel algorithm needs to be distributed. This is straightforward if each agent keeps track of which data are currently unlabeled and reports to other agents only whether yet to be unlabeled data became unlabeled. Upstream and downstream messages mention only which shared datum is affected, and the corresponding acknowledgments report only whether new data were unlabeled.

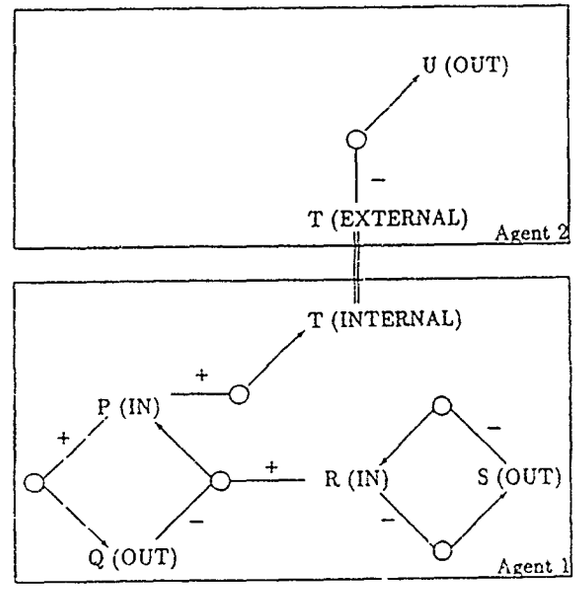


Figure 2: A DTMS network before relabeling

When a group of agents are labeling, their beliefs are in a state not suitable for reasoning. Hence, queries from other agents must be queued until labeling is complete. However, if two agents share a datum, and are involved in separate unlabeling tasks (i.e., initiated by different changes to beliefs), deadlock could occur. Fortunately, the separate unlabeling tasks can be combined into a single one, with a corresponding combination of the subsequent labeling.

3.3 Labeling

Once an area of data is unlabeled, the DTMS must pick candidate labels for the shared data, such that each datum is either OUT in all the agents that share it, or INTERNAL in at least one agent and INTERNAL or EXTERNAL in the rest. Any exhaustive means of picking will guarantee completeness. The following Prolog code shows one means:

```
label-shared([], Shared-labels, Shared-labels).

label-shared([Agent|Agents], So-far, Final) <--
  label-one-shared(Agent, So-far, New),
  label-shared(Agents, New, Final).
```

label-shared relates its first argument—a list of agents—and its third argument—a list of shared data and their labels. The second argument is used to pass commitments about labels for shared data to recursive calls to label-shared. The relation calls label-one-shared, attempting to assign labels to a single agent that are consistent with those already assigned to others. If it finds such an assignment, it recursively attempts to find labels for the other agents. On failure, it backtracks and looks for alternatives to the previous single-agent labeling

This algorithm could be implemented on a fully connected multiagent system by making each agent responsible for generating labels that are consistent with others already generated, as in label-one-shared, and implementing the recursive nature of label-shared with a message passed depth-first from agent to agent. This message needs to contain a list of agents already visited, so that none are revisited, and a list of the already labeled shared data. Also, before an agent passes a message to another, it needs to record its state for

future backtracking. The shared-data labeling fits into the larger labeling process as follows:

```
label(Agents) <--
  label-shared(Agents, [], Shared),
  label-private(Agents, Shared).

label-private([],Shared).

label-private([Agent|Agents],Shared) <--
  local-shared(Agent, Shared, Local-Shared),
  label-wrt(Agent, Local-Shared),
  label-private(Agents, Shared).
```

The private labeling follows the shared data labeling. The private data are labeled one agent at a time. First the relation local-Shared extracts the shared labels relevant to a single agent from the list of all the shared labels Then label-wrt attempts to label the private data consistently with the shared data. Any failure to label causes backtracking. This algorithm will find a Local-and-Shared Consistent and Well-Founded labeling of an unlabeled area in a collection of agents.

Unfortunately, this algorithm has poor computational performance If there is no consistent labeling of the agents, the DTMS will generate all shared data labelings and attempt to label each privately. The performance can be improved by interleaving the labeling of the shared data and the private data Failure in a single agent to find a private labeling consistent with the labels of the shared data will then cause earlier backtracking

```
label(Agents) <-- label-internal(Agents, []).

label-internal([], Shared).

label-internal([Agent|Agents], So-far) <--
  label-one-shared(Agent, So-far, New),
  local-shared(Agent, New, Local-shared),
  label-wrt(Agent, Local-shared),
  label- nternal(Agents, New).
```

Label-internal could be implemented by a message passed depth-first from agent to agent. This message needs to contain a list of the agents visited so far, and a record of the labels given so far to the shared data

Consider again Figure 2. R, S, P, Q, U, and both Ts have now been unlabeled. Agent 1 chooses labels for T and attempts to label his private data in a consistent manner. If Agent 1 chooses INTERNAL as T's label, he finds there is no labeling of his private data to make T internally justified. A next attempt with EXTERNAL is consistent (with S IN and everything else OUT), and Agent 1 passes his label of T to Agent 2. Agent 2 must then label T INTERNAL, but finds there is no way to label U. Agent 2 then backtracks and Agent 1 tries a final attempt to label T, this time as OUT. This succeeds with S IN and everything else OUT, and Agent 2 can also label T OUT by labeling U IN.

3.4 Optimizations

This DTMS algorithm admits several local optimizations:

1. An agent can forego the labeling of its unlabeled shared data by label-one-shared and instead label everything that is unlabeled with label-wrt. This requires a more sophisticated label-wrt that can generate INTERNAL and EXTERNAL labels for the shared data, as well as IN and OUT labels for the private data.

2. An agent can keep a record of label attempts, caching for each combination of shared data labels whether it succeeded or failed to find a private labeling. A call to label-wrt will first consult the cache, thus avoiding redundant work. Ordering the shared data and then indexing the combinations in a discrimination net seems to be a good implementation for this cache.

3 In the above algorithm, only one agent is active at a time. However, there is something productive that the other agents can do: fill in their label caches by attempting to find private labelings for shared data combinations not yet examined. In fact, this effort could be guided by other agents. If agent 1 shares data with agents 2, 3, and 4, when agent 1 passes a label-internal message to agent 2, it could advise agents 3 and 4 about its decisions on the labels of shared data. Then other agents could work only on that portion of their caches that are consistent with agent 1's decision.

4 Discussion

There have been many other attempts to develop systems of cooperating agents or knowledge sources. Early attempts, based on the blackboard model, involved agents with independent knowledge bases. The independence was achieved by restricting agent interactions to modifications of a global data structure—a blackboard—and by minimizing overlap in the agents' knowledge. Later systems allowed richer agent interactions and overlapping knowledge, but the agents were required to have consistent knowledge and to reason monotonically. This led to representational problems, because different experts in the same domain often have different perspectives and *conflicting* knowledge, making it difficult to construct a coherent problem-solving system for that domain. Earlier solutions were to allow inconsistent knowledge bases; this enabled the conflicting knowledge to be represented, but it did not confront the problem of how to resolve the conflicts.

Other researchers have explored negotiation as a means to mediate among conflicting agents. These systems have involved either monotonic reasoners, such as [Sycara 1989], or nonmonotonic, but memoryless, reasoners, such as [Zlotkin and Rosenschein 1989], i.e., reasoners that simply discard old solutions and re-solve in the face of conflicts.

Another approach is to consider the research efforts in multiple-context truth-maintenance systems [de Kleer 1986, Martins and Shapiro 1988] from a distributed viewpoint. These systems manipulate belief spaces, or contexts, in order to remove inconsistent ones. One might imagine each belief space represented by a different agent, who then maintains it. However, in this model, the belief spaces themselves do not interact and, in fact, the belief revision system treats each space separately.

A notable exception to this is the work of [Mason and Johnson 1989], who developed a distributed assumption-based TMS. In this system, agents interact by exchanging data, with their associated assumption sets, and NOGOODS, i.e., bad assumption sets. The agents maintain separate belief spaces and may disagree about an exchanged datum. The agents therefore have Local-and-Shared Well-Foundedness, but only Local Consistency.

The system we presented herein, although an improvement in that it achieves Local-and-Shared Consistency, nevertheless suffers from several deficiencies:

- First, by not supporting some form of explicit negation or reasons for disbelief in a datum, the system allows an agent with less information to dominate an agent with more. For example, if two agents each have an identical justification for belief in a shared datum, and one agent learns a fact that invalidates its justification, the other agent's still-valid justification will be sufficient for both to continue believing in the datum.
- Second, our algorithm can involve significant computational overhead if the agents have shared a large amount of data, if the data have many connections to the rest of the agents' belief networks, and if the status of their beliefs changes frequently.
- Third, we believe unsatisfiable circularities are more likely in a distributed system.

We are currently investigating the likelihood and severity of these deficiencies in real-world application domains. We are also developing a mechanism for negotiation that uses the beliefs supplied by our DTMS.

The above algorithm has been implemented in the *RAD* distributed expert system shell, which includes a framework within which computational agents can be integrated. RAD is a first step toward cooperative distributed problem solving among multiple agents. It provides the low-level communication and reasoning primitives necessary for beneficial agent interactions, but it does not yet guarantee successful and efficient cooperation. The next steps will require increased intelligence and capabilities for each agent, resulting in more sophisticated agent interactions occurring at a higher level.

References

[de Kleer 1986] Johan de Kleer, "An Assumption-Based TMS, Extending the ATMS, and Problem Solving with the ATMS," *Artificial Intelligence*, vol. 28, no 2, March 1986, pp 127-224

[Doyle 1979] Jon Doyle, "A Truth Maintenance System," *Artificial Intelligence*, vol. 12, no. 3, 1979, pp. 231-272.

[Gasser and Huhns 1989] Les Gasser and Michael N. Huhns, eds., *Distributed Artificial Intelligence, Volume II*, Pitman Publishing, London, 1989.

[Huhns 1987] Michael N. Huhns, ed., *Distributed Artificial Intelligence*, Pitman Publishing, London, 1987.

[Martins and Shapiro 1988] Joao P. Martins and Stuart C Shapiro, "A Model for Belief Revision," *Artificial Intelligence*, vol. 35, no 1, May 1988, pp. 25-79.

[Mason and Johnson 1989] Cindy L. Mason and R. R. Johnson, "DATMS A Framework for Distributed Assumption Based Reasoning," in [Gasser and Huhns 1989], pp. 293-317

[McAllester 1980] David A. McAllester, "An Outlook on Truth Maintenance," AI Memo No. 551, Artificial Intelligence Laboratory, MIT, Cambridge, MA, August 1980.

[Russinoff 1985] David M. Russinoff, "An Algorithm for Truth Maintenance," MCC Technical Report No ACA-AI-062-85, Microelectronics and Computer Technology Corporation, Austin, TX, April 1985

[Sycara 1989] Katia Sycara, "Multiagent Compromise via Negotiation," in [Gasser and Huhns 1989], pp. 119-137.

[Zlotkin and Rosenschein 1989] Gilad Zlotkin and Jeffrey S. Rosenschein, "Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains," *Proceedings IJCAI-89*, Detroit, MI, August 1989, pp. 912-917.

Co-operative Issues in a Multi-Agent Vision System

D.M. Lane, A G. McFadzean
Heriot-Watt University
Dept. of Electrical & Electronic Engineering
31-35 Grassmarket
EDINBURGH EH1 2HT
SCOTLAND

Tel: +31 225 6465
Fax +31 225 1137
e-mail: agm@ee.hw.ac.uk
Telex: 727918 IOEHWU g

1. Introduction

This paper describes an implementation of a distributed AI system [1][2][3] which consists of a number of knowledge-based systems cooperating to solve an overall problem. The problem domain is image interpretation where the image contains particularly uncertain and noisy data. A knowledge-based vision implementation was chosen to allow the processing of this poor data. Previous experience with knowledge-based vision has shown that a single agent system is too slow and therefore a distributed system was chosen in an attempt to increase the speed of processing.

The paper builds on previously reported work discussing the system architecture [4][5], by considering some further issues on cooperation, and by presenting some results regarding the performance of the architecture. To this end, a brief description of the overall system is first presented, showing the organisational structure of the individual knowledge-based system agents and also their internal structure. The nature of the cooperation between the agents is then described in more detail, outlining the issues and problems encountered during the system implementation. Results of the system are then given which show cooperation is necessary if a correct solution is to be obtained. Some performance figures in terms of processing speed are also given, showing the reduction in processing time obtained by utilising a multi-agent system. Finally some conclusions and future directions to the work are given.

2 Distributed AI Architecture

The architecture was designed by selecting a suitable distribution for the problem. This was achieved by examining the data abstraction levels of the vision processing task. This firstly involves various types of image processing routines to filter, segment and detect objects in the image. Once individual objects are detected, several features can be calculated for the object (e.g. area, brightness). These tasks all involve processing image pixel data and have therefore been kept within one processing level. Once the features have been extracted, they are stored as a vector. Individual vectors may require further processing but at this level no image pixel data is required. Feature vector processing is therefore a separate second processing level. A third level takes the processed feature vectors and attempts to classify the objects [6]. The domain currently being examined processes images from an underwater sonar device (see figure 5). Objects to be identified are typically divers, remotely operated vehicles, pier legs, anchor chains, and other man-made objects. This three level split functionally decomposes the processing system.

To achieve an even load balance between the processing levels, more processing agents are needed at the lower abstractions, particularly the image processing level. The architecture therefore has a pyramidal shape, as shown in figure 1, with most of the agents at the bottom of the architecture tapering to a single agent, for the classification stage, at the top. The architecture is described in more detail elsewhere [4][5].

Within each level of the architecture, the processing agents are the same, each agent differing only in the data they process. However each level contains differing knowledge to process their data

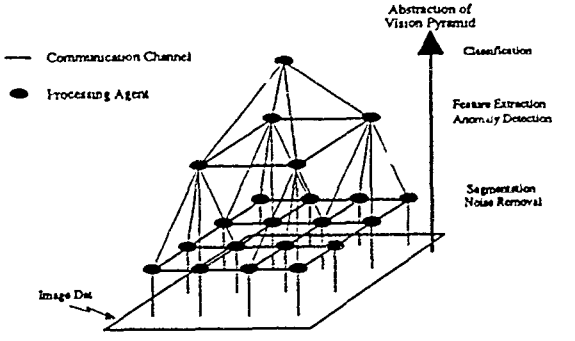


Figure 1: Distributed AI Architecture

type. A common communications protocol is used between the agents to allow differing internal architectures to be used. Currently all the agents are blackboard based [7] (building on work in [8][9]) with additional features added to allow for communications with other agents and also to allow for pre-emptive scheduling of knowledge sources for fast response to external stimuli.

To achieve the physical distribution of the system, the architecture has been implemented on a transputer based multi-processor system. Individual agents are implemented using a number of multi-tasking processes. This allows for responsiveness to external stimuli. Communication between the multi-tasking processes and between agents is achieved using message passing. Figure 2 shows a simplified data-flow diagram for a single agent.

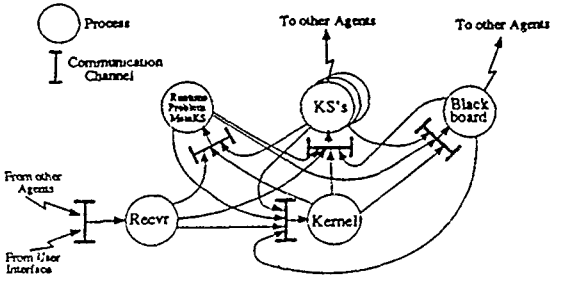


Figure 2: Simplified Data-flow for an Agent.

The primary motivation for the distribution of the system was to increase the speed of processing whilst maintaining some generality in the system design. However the distribution, and resultant cooperation required between agents, has raised a number of issues and problems. These issues are dependent on the processing level within the pyramid. Issues for the image processing levels and feature vector levels are therefore described in the next two sections.

3. Issues in Cooperation at the Image Processing Level

At the image processing level in the pyramidal structure, cooperation is required for two reasons. Firstly cooperation can be used to reduce uncertainty between agents in processing results. Secondly cooperation (in the form of exchanging partial results between agents), is necessary if agents are to arrive at a correct solution.

The issue of reduction of uncertainty can be of two types. Within the blackboard of each agent, all data and hypotheses have an associated certainty value. This value is propagated and modified by agents as hypotheses are inferred and communicated to other agents. This form of uncertainty can be reduced using cooperation by, for example, two agents producing the same result from the same data. The fact that two independent agents infer the same result boosts the confidence for that result. However, this form of uncertainty reduction is not the main use of cooperation.

Within the image processing level, individual agents process small sections, or blocks, of the complete image. Each treat their blocks as a small image and process it independently. However if agents do not cooperate with one another, the selection of image processing parameters between adjoining agents may be different. This will result in discontinuities in detected objects across the boundaries between adjacent blocks. This can result in objects straddling the boundary having large discrepancies in their features as shown in figure 3. These discrepancies in turn lead to high uncertainty in object features.

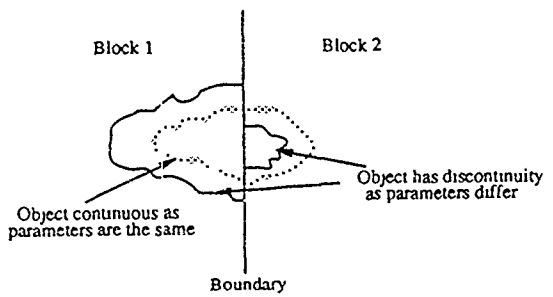


Figure 3: Discontinuity at a Boundary

The use of cooperation between agents in the selection of parameters can remove the uncertainty by agents agreeing on parameters across the boundary. However cooperation is not simple because blocks within a image can have up to eight adjoining blocks. The agreement of one agent with another eight agents who may each in turn be cooperating with a further eight agents, becomes a large interdependent negotiative task. In practice it is simpler if agents cooperate with neighbours individually, agreeing on an image processing parameter between these two agents' boundaries. The image processing parameter is then modified linearly from boundary to boundary in a manner similar to the way shading is calculated at polygon edges in computer graphics. This ensures that no discontinuities exist at any boundaries and also breaks the cooperation task down into a number of independent interactions. This is an example of the issues of cooperation resulting in a modification of the processing methods.

In addition to cooperation involving processing parameters, it is necessary to cooperate by exchanging pixel data between agents. In many image processing algorithms, individual pixels are modified depending on the value of their neighbours. This causes irregularities at image edges due to edge pixels not having neighbouring pixels surrounding them on all sides. If this effect was to be ignored in the distributed system, irregularities would occur at block boundaries and discontinuities would be created between blocks. To remove this problem, agents must cooperate by exchanging pixel data at the edges of their blocks. The number of pixels exchanged is dependent on the number of

neighbouring pixels being examined by the image processing routine. Typically this is a strip of between one to ten pixels.

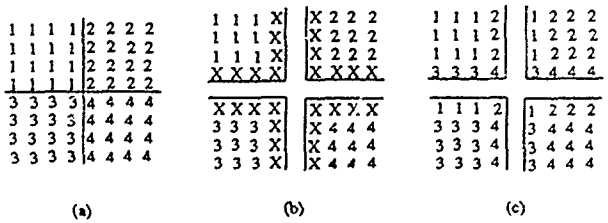


Figure 4: Image Overlap Cooperation
(a)original image (b) distributed blocks with blank overlaps
(c) blocks after cooperation

Figure 4 shows the cooperation for an overlap of 1 pixel. Figure 4a is the corner of four blocks within the original complete image. Figure 4b shows the blocks as they are given to each agent, together with a one pixel blank overlap surrounding the block. Figure 4c shows the blocks after cooperation has been completed. It can be seen that each blank overlap has been filled with data from the other three blocks. This boundary or overlap around each agent's block allows the agent to correctly process the pixels within the block size it was originally allocated. This form of cooperation must be performed before each image processing routine to allow the boundaries to be kept continuous.

The image processing level has shown that it is not only necessary for cooperation of data, but also of processing parameters if a correct solution is to be obtained. This second form of cooperation reduces uncertainty within the solution while still allowing the data to be processed in a localised manner. This results in a greater 'fine tuning' of the processing to localised areas of the image. The solution quality of the cooperating system is therefore higher than that obtained by a single processing agent.

4. Issues in Cooperation at the Feature Vector Level

The feature vector level receives feature vector data for each of the objects found by the image processing level. However some objects may have been split over a number of image processing agents and are therefore passed to the feature vector level as partial objects. Partial objects can only have a partial object feature vector calculated. This is because a feature vector for the complete object requires all the object data before it can be calculated. It is the job of the feature vector level to ensure that all partial objects are combined and a complete object feature vector calculated for each object. The completed objects and their feature vectors are then passed onto the single agent in the classification level.

A single feature vector agent receives objects from several image processing agents. On receipt of an object, the feature vector agent checks if the object is a partial object, and if so a test is made to see if this agent is likely to have more data for the same object. This is the case if two image processing agents output their data to the same feature vector agent. If this is the case the partial objects and their feature vectors are combined.

Note that partial object feature vectors cannot be converted to a full object feature vector, at this stage, as an agent has no way of knowing how may parts an object was originally divided into. Conversion to complete object feature vectors can only take place when all data from all the agents below has been received and, where applicable, combined. This ensures that all parts of an object will have been combined before conversion to a complete object feature vector. If an image processing agent does not find any objects it must still therefore send a message indicating that no object data will be sent.

Objects can also be split over several feature vector agents. In this case, cooperation must be used to combine the partial objects. The feature vector agents must first establish if they contain several parts of an object themselves, and if so combine these before passing the partial object to a neighbour.

Cooperation within this level is used to establish where all the parts of an object are and to decide where they should all be sent. Agents initially have no knowledge of the number of parts an object has been split into, or where most of these parts are. They know only which of their immediate neighbours contain more parts of the object. To establish if any more parts to the object exist, each agent asks its immediate neighbours, known to contain further parts of the object, which of their neighbours also contain further parts. These agents, in turn, are asked until a complete list of agents containing parts of the same object is built up. This list is constructed in each agent containing part of the same object. Cooperation has thus been used to establish where all the data is stored within the network.

Negotiation between all the agents containing the object is then used to establish which agent will collect all the object parts. The choice of this agent is made by examining the processing loads of the processor each agent is running on and selecting the most lightly loaded. Individual agent load's are not used as more than one agent may be running on one transputer processor. Once the collecting agent has been established, each agent holding other parts of the object sends its part to the collecting agent. This agent then combines the parts. Once all the parts have been combined, the partial object feature vector is converted to a feature vector for a full object. This is then passed up to the agent used for classification.

5. Results

This section will show some typical results of processing an image with the distributed system and compare the results to those obtained by a single agent system.

A typical sector-scan sonar image is shown in figure 5. The image contains a number of objects (a diver, an anchor chain and pier piles are the three largest objects). This image was processed by a pyramid network of 16 image processing agents, 4 feature vector agents and a single classification agent. The result of processing by the image processing level is shown in figure 6. Here, the image has been split up into 16 blocks as shown by the grid lines. Each image processing agent received one of the blocks as input. The objects detected within each block have been boxed. Two objects have been split between image processing agents. One object has been split between four image processing agents, the other between two agents.

Figure 7 shows the result after the feature vector stage has been run. The grid lines indicate that four feature vector agents have been used (these each have four image processing agents inputting data to them) Both objects have been successfully combined, with one object (the smaller) being combined by one feature vector agent, the other object requiring cooperation between two feature vector agents.

Figure 8 shows the output of the image processing level when the cooperation of image overlaps is turned off. It can be seen that the objects straddling the boundaries between agents have been incorrectly processed. This results in the classification stage producing the wrong classifications for these objects. Currently cooperation of parameters is not used as all image processing agents use the same parameters.

The overall speed of the distributed system is considerably faster than previous times obtained from a single agent system. Implementation was originally carried out on a Sun 3/160 workstation where a single agent took approximately 5 minutes to process an image. Porting this system to a T800 transputer where images are stored in memory rather than on disk, resulted in processing times of around 102 seconds for a single image. Note that the times quoted here do not include the classification stage as in all systems this is a single agent running on the Sun. The distributed system of 16 image processing agents and 4 feature vector agents takes approximately 19 seconds to process an image giving approximately a five fold increase in performance.

The cooperating system's time is not as fast as would initially be expected with a 19 fold increase in processing power. This is in part due to poor load balancing between the processing levels. With 16 processors the image processing level takes 15 seconds compared with 4 seconds for the feature vector

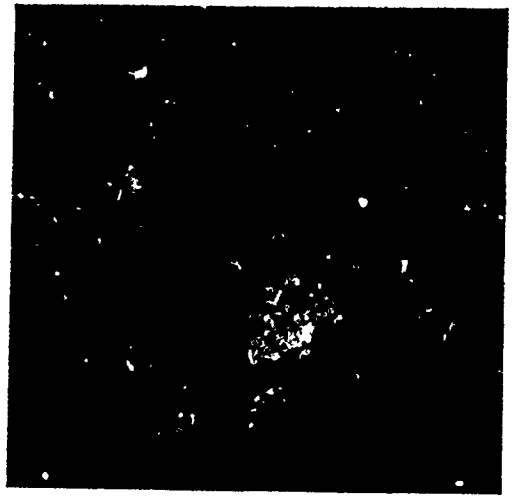


Figure 5: Typical side-scan sonar image

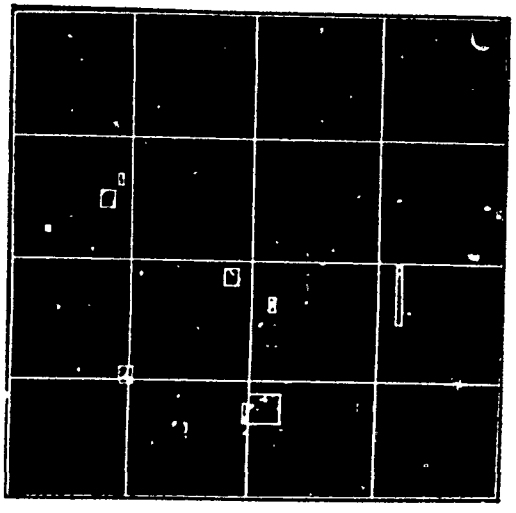


Figure 6: Result of Image Processing Level

level. For the single agent case, the image processing takes approximately 101 seconds (only 1 second is needed for the feature vector level as it has no partial objects). The 16 processors do not process in one sixteenth of this time (around 6.3 seconds) because of the extra processing due to the overlaps For a 4x4 grid of agents, processing a 256x256 image, each agent processes a 64x64 pixel block. However

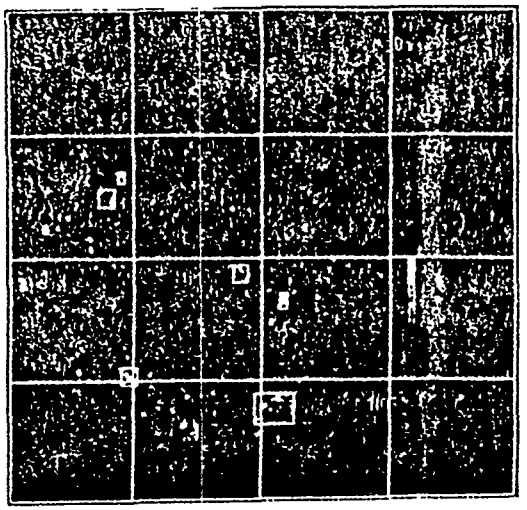


Figure 7: Result of Feature Vector Level

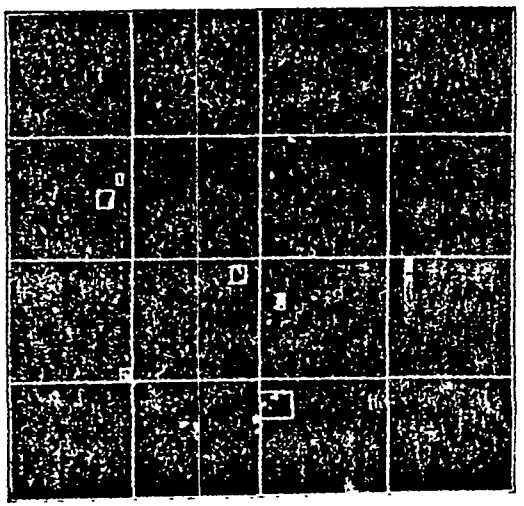


Figure 8: Result of Image Processing with no Cooperation

overlaps averaging 5 pixels are used resulting in 74x74 pixel blocks This increases the processing by approximately 33%. An image is processed by at least five routines giving a 168% increase in total This increases the ideal 6.3 seconds to 16.9 seconds In practice, some overlaps are less than 5 pixels and therefore the overall time is slightly less than 16 9 seconds at 15 seconds

The above results show that cooperation between the agents does not incur much overhead within the system However the results do show that distributing some systems can mean large amounts of duplicate processing is necessary This means that processing speed gains are not as great as initially anticipated and will have a limit (in our application this occurs when the overlap size is larger than the block size).

6. Conclusions and Future Work

The prime motivation for building a system of cooperating knowledge-based systems has been to increase the speed of processing whilst maintaining some generality in the architecture This has been successfully achieved by the current system.

The distributed system can also give improved solutions over the single agent system by making use of fine tuning to localised areas. It has also been shown that cooperation is a necessary undertaking if a correct solution is to be obtained. Some of the issues of the cooperation between the knowledge-based system agents and how these can result in modification of processing methods have also been discussed.

Currently we are investigating the inclusion of additional real-time features into the architecture [10] and assessing the impact of a fault tolerant requirement on the overall system design. Both are critical attributes if such systems are to be used in practical robotic applications.

7. Acknowledgements

Thanks are due to Euan Robertson who participated in the initial inception of the pyramidal architecture idea, and to Mike Chantler who has contributed to regular group discussions on distributed artificial intelligence.

This work was jointly funded by the Marine Technology Directorate Ltd, agents of the UK Science and Engineering Research Council, BP Petroleum Development Ltd, Shell UK Exploration & Production, Ferranti ORE Ltd, Honeywell Inc (USA), Department of Energy - Offshore Supplies Office, and the Ministry of Defence - Admiralty Research Establishment.

8. References

[1] Durfee E.H., Lesser V.R., "Using Partial Global Plans to Coordinate Distributed Problem Solvers", *Proc. 10th Int Joint Conference on Artificial Intelligence*, Vol. 2, 1987, pp 875-883.

[2] Lesser V R., Corkill D.D., "Functionally Accurate, Cooperative Distributed Systems", *IEEE Trans on Systems, Man, and Cybernetics*, Vol SMC-11, No 1, Jan 1981, pp.81-96.

[3] Decker K.S., "Distributed Problem-Solving Techniques: A Survey", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-17, No.5, September/October 1987, pp.729-740

[4] Lane D.M., Chantler M.J., McFadzean A G., Robertson E W., "A Distributed Problem Solving Architecture for Knowledge Based Vision", Distributed Artificial Intelligence Vol II, Ed: M Huhns, Morgan Kaufman, 1989. Also in *Proc 1988 AAAI Workshop on Distributed Artificial Intelligence*, Lake Arrowhead, Ca., May 22-25, 1988.

[5] Lane D M, Chantler M.J., McFadzean A.G, Robertson E.W., "A Distributed Problem Solving Architecture for Sonar Image Interpretation", *Journal of the Society for Underwater Technology*, Vol. 14, No. 3, Autumn 1988, pp 12-18

[6] Stoner J.P., "Knowledge Base Design for Artefact Classification in Sector Scan Sonar", Final Report, Automation of Subsea Tasks Managed Programme of Research 1988/90, Heriot-Watt University, 1990

[7] Nii H.P., "Blackboard Systems: The blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", *AI Magazine*, Summer 1986, pp.38-53.

[8] Lane D M., "The Investigation of a Knowledge Based System Architecture in the Context of a Subsea Robotic Application", *PhD Thesis*, Dept. of Electrical and Electronic Engineering, Heriot-Watt University, 1986.

[9] Russell G.T., Lane D M., "A Knowledge-Based System Framework for Environment Perception in a Subsea Robotics Context", *IEEE Journal of Oceanic Engineering*, Vol. OE-11, No. 3, July 1986, pp 401-412.

[10] Chantler M.J., Lane D M., McFadzean A G., "Responsive and Time-Constrained Reasoning in Autonomous Vehicles", *Proc. 1989 International Conference on Systems, Man, and Cybernetics*, Cambridge, Ma., Nov 14-17, 1989, Vol. 3, pp.566-567.

A Multi-Agent System for Dual-Arm Kinematic Motion

D.M. Lane, A.W. Quinn
Heriot-Watt University
Dept. of Electrical & Electronic Engineering
31-35 Grassmarket
EDINBURGH EH1 2HT
SCOTLAND

Tel: +31 225 6465
Fax: +31 225 1137
e mail: awq@ee.hw.ac.uk
Telex: 727918 IOEHWU g

Abstract

This paper introduces a possible solution to the problem of resolving kinematic motion in a dual-arm system by the use of cooperation and collaboration within a multi-agent environment. A general multi-agent system has been developed using the MUSE AI Toolkit, and this system configured to model two manipulator arms. A single agent represents each joint in the arms. Different strategies are employed by these joint-agents to enable the end-effectors reach a specified goal. This approach to resolving kinematic motion resulted in the end-effectors successfully reaching their goal, with further work being carried out in developing strategies to allow obstacle avoidance and collision-free motion to take place. It is hoped that this approach will result in a system which can cope with the problem of dual-arm cooperation more readily than existing systems. The main issues addressed in this paper are dual manipulator cooperation, multi-agent environment; and behavioural strategies.

1. Introduction.

Before this solution to resolving kinematic motion is examined, the multi-agent environment is described in terms of its basic component the single-agent cell, and it is shown how a number of these cells are interconnected to produce a *suitable configuration for the dual-arm manipulator problem.*

In this configuration, the multi-agent system would receive the desired goal point of the end-effectors from a planner with some additional constraints such as the orientation of the end-effectors gripper, for example. The multi-agent community would then cooperate to produce an output consisting of a series of joint angles for both manipulators to ensure collision-free motion of the arms and the avoidance of external obstacles within the manipulators workspace.

The long term objective of this project is the realisation of an autonomous dual manipulator arm system mounted on an underwater vehicle. This vehicle would be capable of performing various tasks, such as the inspection and repair of deep-water pipelines, without the need for human-operator interaction. The arms being incorporated into this project are the TA9 Manipulators constructed by Slingsby Engineering Ltd (fig 1). These arms are already being used in underwater environments, though at present they are mainly operator driven.

This paper concentrates on the more short term objective of resolving the kinematic motion of the *dual-arm system*, though related projects include the planning and closed loop control for the arms, sub-sea vision, image segmentation and interpretation, and investigation into a computer architecture capable of integrating all these sub-systems together [1],[2],[3].

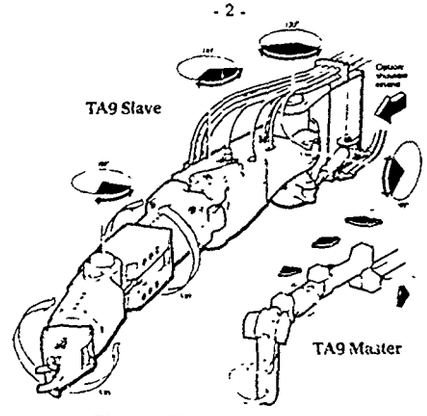


Figure 1: TA9 Manipulator Arm.

2. *Existing Systems.*

Previous work in this area has concentrated on finding the inverse solution to kinematic equations [4], and implementing Brook's Subsumption Architecture [5]. These systems encountered difficulties due to complexity in the inverse kinematics method and scalability in the case of the Subsumption Architecture.

The multi-agent system differs from previous methods of controlling manipulators by using a community of simple agents with limited communication channels which cooperate together to reach a solution. This approach to problem solving using interacting agents was first described in [6].

3. Multi-Agent System.

The following section describes the multi-agent system in terms of its basic component the single-agent cell, and how this cell is interconnected with others to produce a multi-agent community which could be used to implement various problem solving techniques. Though this multi-agent *environment could be applied to many areas, this paper concentrates on how such a system can be suitably configured for solving the dual-arm manipulator problem.* This section also describes the various actions or behaviours that an individual agent can adopt to aid the overall system in reaching the goal state.

3.1. *Single-Agent Cell.*

The multi-agent system is built from a number of simple single-agent cells (fig.2). These cells are identical in nature and internal construction with the exception of a few variables which characterise and identify the individual cells.

Each cell, or agent, only *communicates with a limited number of other agents* which, for these initial experiments, is three. Therefore, the only information an agent has access to is limited to the agents it can talk to, otherwise known as its acquaintances. This implies that no single agent can hold a *complete picture of the global state of the system* or have a full understanding of the consequences of its actions.

Each agent performs various actions in order that a common goal can be achieved without having to know about the total size of the system, *its configuration*, or how the *other agents interact*. This leads to an essentially simple system where a large community can be constructed quickly and easily, keeping the knowledge contained within any one agent to a minimum.

There exists within each agent a local memory function which can store a *few relevant variables*, allowing the agent to keep track of previous actions or moves. Agents can only interact by messages

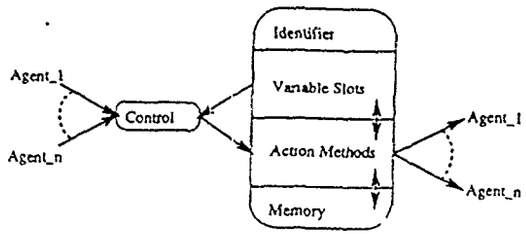


Figure 2: Single-Agent Cell.

passing and as mention already there is no shared memory. In contrast to the ECO Problem Solver [6], agents have the ability to directly manipulate the states of other agents, manipulating them in the sense that the altered state is the consequence of the dominating agents action.

As with ECO, certain principles hold true for this particular system The first of these is that the behaviour of an agent is decided upon by the condition of its own state and the information contained in the messages it receives from other agents Also when deciding between a set of possible actions the agent will choose the action which minimises a defined cost function However this is not the only criteria which determines the course of actions taken and is not essential to the operation of the system.

3.2. Multi-Agent Community.

The TA9 manipulator arms have seven degrees of freedom and the most obvious method of modeling such a system using the single-agent cells is to have one agent represent each joint in the arm This approach fits in well with the constraint of limited communication channels since each joint n the arms has only two immediate neighbours (fig 3).

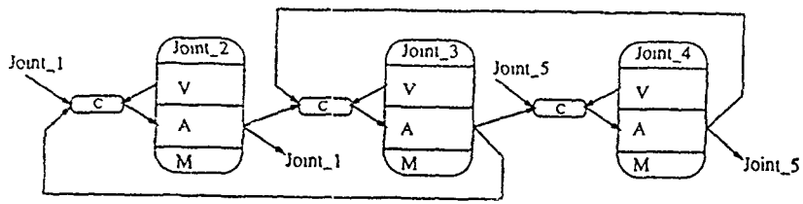


Figure 3: Three Interconnected Agents.

In addition to these agents, there also exists an agent which oversees the system. The main role of this "supervisor" is to simply observe and detect oscillations if and when they occur. Oscillations occur when the individual agents unsuccessfully and repeatedly go through the same action sequence in their attempt to reach the goal point When an oscillating situation is detected, the supervisor agent forces the individual joint agents to adopt a different strategy in order to break the system free of its repetitive loop

Each joint agent holds the characteristics of the particular arm joint which it represents, containing its present position in space, its maximum and minimum angle movements, the acquaintances to which it can talk, memory stacks to record previous movements amongst other variables. Each agent *also has access to a library of functions which dictate the various types of actions it can perform* (fig 4).

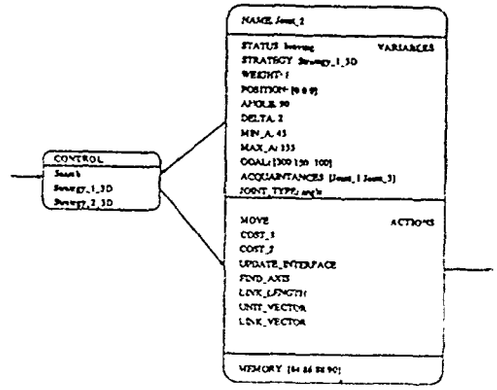


Figure 4: Example of a Single Joint Agent.

Whenever an agent decides to alter the value of its joint angle this affects the position in space of the other agents. Instead of there being a global picture of the arms configuration somewhere in memory which is altered using transformation matrices, each agent only knows its own position in space and can ask for the positions of the adjacent joints to which it can communicate. Using this information, the moving joint broadcasts its position in space with an axis of rotation to its acquaintances *which in turn propogates the information to the relevant joints. These then calculate their own updated* position in space.

The way in which an agent moves its joint angle is determined by the particular strategy which it is following. There are two basic types of strategy, these being behavioural strategies and search strategies

A major difference exists between the behavioural strategy approach and the search strategy when viewing the resulting physical motion of the arm. The behavioural method gives motion while a solution is being calculated, whereas the search method reaches the required solution before it begins to move.

The following two sections describe the other differences between behavioural and search strategies.

3 3 Behavioural Strategies.

Each individual joint agent can adopt a particular behavioural strategy, which may vary according to the current state of its variables. The agent chooses a strategy which it thinks will help the system, and the end-effector agent, to reach the goal point.

One, very simple, behavioural strategy tries to minimise the distance between the end-effector and goal point whilst disregarding what other agents are doing. Each agent changes the angle of the joint it represents and then calculates the new distance between the end-point of the arm and the goal point. If this distance decreases then the change in angle is made permanent, otherwise the move is reversed and the joint angle is moved in the opposite direction for the next move

Another, more sophisticated strategy, attempts to make the end-effector follow a specified trajectory towards the goal point. The trajectory with which the system has been tested is a straight line lying between the starting position of the end-point and the goal point. This strategy has two conditions which need to be met before making an angle move permanent. The first is the same as that for the simple strategy in that the end-point must have moved closer to the goal. The second condition is that the perpendicular distance between the end-point and the straight line trajectory must not exceed a predefined value.

The simpler behavioural strategy results in a "groping" type motion of the arm as it manouvers its way towards the goal point. The trajectory following strategy gives a smoother motion though is computationally more expensive. Both strategies suffer from the problem of local minimas in their cost functions resulting in the arm being unable to reach the end-point. These cases tend to occur when the desired goal point is close to the perimeter of the working envelope of the arms i.e. the set of possible joint angles which give a correct result is at a minimum.

3.4. Search Strategies.

The result of a search strategy is that a specified area of the manipulator's workspace is exhaustively searched to find the best joint configuration which satisfies the goal state. Search strategies are a function of the supervisor agent where it dictates to each individual agent the angle moves required to carry out the search. Once the most satisfactory configuration is found, the arms move their joints accordingly to reach this new position.

Using search strategies has two basic purposes. The first reason is for comparison with the behavioural strategies, to check which method reaches the desired goal point quicker. The second purpose is to see how the final configuration of joint angles differ from that of the behavioural strategies.

4. Results and Future Work.

Initial experiments have compared the performance of the simplest behavioural strategy against the performance attained by the search strategy algorithm. The results show that the behavioural strategy, even in its most simplest form, gives an improvement in the time taken for a single manipulator arm to reach a specified goal point. Using a search strategy in 3D space is expensive in terms of computing time and also cannot take consideration of obstacles in the manipulators workspace.

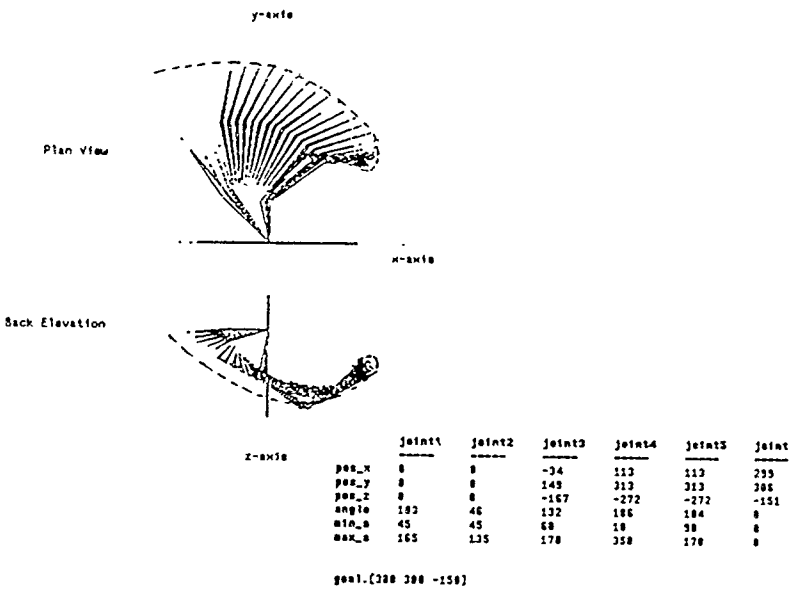
Figure 5 shows an example run of the system with all the agents following the simplest behavioural strategy. The dotted line shows the path taken by the end-point of the arm and it can be seen that this point overshoots the goal slightly before correcting its path to reach the desired end position shown by the cross.

Further work will involve developing alternative behavioural strategies to improve upon the results already obtained. These alternative strategies will be designed to cope more readily with the problem of obstacle avoidance, and also to investigate the problem of local minimas in the strategies cost functions.

Further work will also include the development or use of a system which incorporates the more conventional inverse kinematics approach in order to compare these two methods. This needs to be done to determine which method is the quicker to reach a solution in the presence of obstacles and moving objects such as another manipulator arm.

At present the TA9 manipulators are being simulated in three dimensions on a Sun workstation, using Muse [7] as the development toolkit. Work is also being carried out on the integration of the multi-agent system with the actual TA9 manipulator arms.

Future work also involves the transfer of the existing configuration, which is simulated on a sequential machine, onto a suitable parallel system. Initial investigation has been carried out in identifying such a suitable environment and one possibility might be a transputer based architecture under development [1].



	joint1	joint2	joint3	joint4	joint5	joint
pos_x	0	0	-34	113	113	255
pos_y	0	0	149	313	313	305
pos_z	0	0	-167	-272	-272	-151
angle	183	46	122	186	184	0
min_a	45	45	60	10	90	0
max_a	165	135	170	350	170	0

Figure 5: Example Run of the Simple Behavioural Strategy.

5. Conclusions.

This multi-agent configuration for resolving the kinematic motion in a dual-arm system is being developed as an alternative to the accepted standard practice of using inverse kinematics. Current techniques have taken on board this standard method and attempted to integrate this with path planning and collision avoidance algorithms, resulting in a system which is computationally expensive.

This alternative approach of using a multi-agent community can be seen as an attempt to address the problem of collision-free motion and integrating it with a different approach to modeling and resolving the motion of a manipulator arm. Objects known to be adjacent to the manipulator will influence the behaviour of the agents controlling joints which could cause a collision. It is hoped that this will eventually lead to a system which can cope with the problem of dual-arm cooperation more readily than existing systems.

The multi-agent community can also be considered as more adaptive, since various different behavioural and search strategies can be employed whilst the arms are in motion. This environment will also be receptive to dynamic changes in internal and externally imposed constraints upon the system. It will be able to accept alterations to the current plan and produce the kinematic motion necessary to enable the arms to reach their new goal state without backtracking to the original positions.

As well as being an alternative to conventional approaches, the multi-agent system will eventually have the capability to reach solutions with greater speed. This is due to the attainable level of parallelism possible with independent agents communicating over low bandwidth channels.

initial results have shown that the multi-agent behavioural approach already compares favourably with existing search methods even though the essentially parallel community is being simulated on a sequential machine. Once a fully developed parallel environment is created then a further improvement in system performance may be possible

6 Acknowledgements

This work was jointly funded by the Marine Technology Directorate Ltd. agents of the UK Science and Engineering Research Council, BP Petroleum Development Ltd, Shell UK Exploration & Production, Ferranti ORE Ltd, Honeywell Inc (USA), Department of Energy - Offshore Supplies Office, Ministry of Defence - Admiralty Research Establishment, and Slingsby Engineering Ltd

7. References

[1] Lane D.M., Chantler M.J., McFadzean A.G., Robertson E.W., "A Distributed Problem Solving Architecture for Knowledge Based Vision" Distributed Artificial Intelligence Vol II, Ed. M. Huhns, Morgan Kaufman, 1989. Also in *Proc 1988 AAAI Workshop on Distributed Artificial Intelligence*, Lake Arrowhead, Ca., May 22-25, 1988

[2] Lane D.M., Chantler M.J., McFadzean A.G., Robertson E.W., "A Distributed Problem Solving Architecture for Sonar Image Interpretation", *Journal of the Society for Underwater Technology*, Vol. 14, No 3, Autumn 1988, pp 12-18.

[3] Stoner J.P., "Knowledge Base Design for Artefact Classification in Sector Scan Sonar", Final Report, Automation of Subsea Tasks Managed Programme of Research 1988/90, Heriot-Watt University, 1990.

[4] Koivo A.J. "Fundamentals for Control of Robotic Manipulators", Wiley Press, 1989.

[5] Connell J.H., "A Behaviour-Based Arm Controller", *IEEE Transactions on Robotics and Automation*, Vol. 5, No 6, Dec 1989, pp 784-791.

[6] Ferber J., "ECO Problem Solving. How to Solve a Problem by Interactions", *Proc 9th Workshop on Distributed Artificial Intelligence* Rosario Resort, Eastsound, Washington, Sep 12-14, 1989

[7] MUSE - *An AI Toolkit*, Cambridge Consultants Limited 1988

B 04

3 copies.

The MCS multi-agent testbed: experiments and developments

James Doran

Department of Computer Science
University of Essex
Colchester CO4 3SQ
Tel. 0206 872338
Fax. 0206 873598
E-mail doraj@uk.ac.essex

ABSTRACT

The Testbed

This paper describes the MCS multiple agent software testbed which has been developed as a research tool in the University of Essex, Department of Computer Science, and briefly reports experiments using it. MCS is coded in CProlog and NIP and runs in UNIX on Sun-3 and Sun-4 workstations.

The testbed is built around the IPEM planner (Ambros-Ingerson and Steel, 1988) which is named for Integrated Planning, Execution and Monitoring. IPEM provides a simple and well-defined framework in which to integrate these processes. Representation integration is achieved by consistently incorporating execution and monitoring information into the standard non-linear partial plan representation. Control integration is achieved by using a production system architecture where IF-THEN rules, referred to as 'flaws' and 'fixes', specify partial plan transformations, that is, meta-actions. Conflict resolution is achieved by the use of a scheduler which embodies the current problem-solving strategy. Since execution and plan elaboration actions have been designed to be independently applicable, and execution of an action is a scheduling decision like any other, the framework supports interleaving of planning and execution of non-linear hierarchical plans.

The MCS testbed itself provides the facility to specify any (small) number of agents each with its own database of 'beliefs' and operators, the latter comprising action representations in the STRIPS formalism together with associated action execution procedures. Each agent has independent access to the planner and has a set of plans currently being worked upon. Concurrency is simulated by giving each agent in turn a 'flaw-fix' cycle, corresponding to the execution of a single rule in a production architecture. With each agent may be associated simple demons which fire whenever particular propositions appear in the agent's database. The execution part of a demon may be any action or meta- action available to the agent. This provides agents with a form of reactivity.

Communication between agents is achieved by the execution by agents of actions (planned or reactive) whose execution procedures call inter-agent interface primitives -- which, in effect, write to the target agent's database. Alternatively, communication may be via one or more agents treated as environment agents.

A menu-driven interface is provided for interation with and monitoring of the running testbed.

Experimentation

MCS has been and is being used to support a variety of experimental work. Specifically:

--a small scale implementation of a distributed sensing network for monitoring vehicle movements, based on that designed and studied by the DAI research group at the University of Massachusetts (Sim S K, 1988)

--a study of plan delegation in a multi-agent environment (using a variant of MCS called DePlan) focussing on requests as partially elaborated plans (Hopkins, 1990).

--a study of planning in a distributed problem solving hierarchy required to perform an (abstract) distributed production task. The structure of the hierarchy (including its communication channels) is assumed fixed, and the emphasis is on exploring the consequences for problem-solving of alternative design choices for intra-agent processing, especially the nature, location and degree of abstraction of planning (Li, forthcoming).

--a study of the variation of behaviour in multiple agent distributed problem solving systems when the agents employ various types of social model. By an agent's 'social model' is meant its dynamic representations of other agents and the (sub-) groupings which they form or are believed (by the agent) to form. Social models may be partial or erroneous. Agents' social models in this sense clearly play a pivotal role in action choice (whether 'situated' action or action in consequence of predictive planning) and in inter-agent communication. Limitations or errors have major behavioural implications. Carvajal has studied the trade-off between social model complexity (intuitively, how much an agent knows about other agents and their groupings) and problem solving effectiveness in a community of agents where a pattern of reciprocal exchanges is required for a simple production process (Carvajal, Forthcoming)

--the EOS project. Around 20,000 years ago in Southwestern France a relatively simple hunter-gatherer society rapidly developed enhanced social complexity and a much richer culture, including world famous cave and rockshelter art. The causes and processes of this remarkable and unprecedented social development are obscure. The EOS project (a collaboration between the Universities of Essex, Cambridge and Surrey, which is funded by the Joint Council Initiative on HCI and Cognitive Science) is using the MCS testbed to examine a specific process model for the emergence of human social complexity in the Upper Palaeolithic of Southwestern France put forward by Paul Mellars of Cambridge University. Mellars' model takes into account environmental factors, increasing population density, more complex problem solving for subsistence, and changes in social perception and awareness. The primary objective of the EOS project is to give the Mellars model a consistent computational interpretation and implementation, and to derive its detailed properties by computer based experimentation. The results obtained will help interpret the evidence of the archaeological record.

Issues

Although most of this experimental work is still in progress, certain recurring research issues have emerged. Prominent amongst them is the need to identify the circumstances in which agents being designed to cooperate in distributed problem solving should be designed to engage in predictive planning rather than purely reactive choice of actions. In the past it has often been assumed that complex predictive planning is desirable in the belief that the more powerful the technique always the better. But in practice this assumption leads to inefficiency within the system, and to a potentially unnecessary and irrelevant confrontation with some hard research problems for the system designer. Computational agents should be designed to use predictive planning only when they must and to the degree that they must. In particular we need to find ways systematically to determine at what level of abstraction predictive planning should be carried out when it is used. Action planning will almost inevitable involve some degree of abstraction compared with action execution. How much is an important design variable. We have very little understanding of the trade-offs.

A higher-level issue is the nature of the mechanism which calls and controls the planning process and its siblings within an agent. In MCS this mechanism is arbitrary and trivial. Demon firing always has priority over planning whatever the tasks being performed by the demons. A less arbitrary ways of choosing meta-processes to run is required, ultimately related to the tasks to be performed by the agent community.

Testbed development

The existing MCS testbed is being enhanced in various ways, notably:

--run-time creation and deletion of agents

–heirarchical structuring of agents' social models

–improvement of run-time interface and instrumentation

–improvement of runtime efficiency (using Quintus Prolog)

In addition a 'second generation' testbed successor to MCS is being designed as part of the IED/SERC CADDIE project.

References

Ambros-Ingerson J.A and Steel S (1988) Integrating Planning, Execution and Monitoring. Proceedings of the 1988 Conference of the American Association for Artificial Intelligence, St Paul, Minnesota AAAI, pp 83-88.

Carvajal H (forthcoming) A Computational Implementation of Actors' Social Models Using the MCS/IPEM Software Testbed. M Phil dissertation, Department of Computer Science, University of Essex, Colchester, UK.

Doran J (1990) Using Distributed AI to Study the Emergence of Human Social Organisation. Department of Computer Science Research Memorandum CSM-154, University of Essex, Colchester, UK.

Hopkins C (1989) Meta-Level Planning for Multi-Agent Cooperation Proceedings of AISB 89 Conference, Pitman & Morgan-Kaufmann, pp185-189.

Li Y. Forthcoming MSc dissertation. Department of Computer Science, University of Essex, Colchester, UK

Sim S. K. (1987) A Partial Implementation of the Durfee, Lesser and Corkhill Model M Sc IKBS Dissertation. Department of Computer Science, University of Essex, Colchester, UK.

NEST: A Networked Expert Systems Testbed for Cooperative Problem Solving

submitted to:
the Working Conference on Cooperating Knowledge Based Systems

Michael J. Shaw
Barbara Harrow
Sherri Herman

Beckman Institute for Advanced Science & Technology
University of Illinois at Urbana-Champaign
Urbana, IL 61801
USA

Abstract

The advances in information technology have made computer networks ubiquitous. The increasing uses of distributed processing and electronic meeting systems that involve multiple users necessitate the development of some new principles of information system design that consider the distributed, coordinated nature of group problem solving. This paper describes a framework for designing group problem-solving systems based on distributed artificial intelligence. Among the design issues, we find the coordination mechanisms and the learning schemes used to be of particular imponance. An implementation example of a network of expert systems is used to illustrate the distributed artificial intelligence approach.

1. Introduction

This paper is develops a distributed artificial intelligence (DAI) framework for cooperative problem solving. There are three objectives of this research: (1) Incorporating cooperative, multi-agent problem solving capabilities for DAI systems and describing an implementation of the framework by a network of expert systems. Just as an artificial intelligence system is used to model the problem-solving process of a single agent, so does a DAI system model the group problem-solving process of multiple agents. Besides the inference capabilities necessary for every type of problem-solving agents, a critical component in developing AI-based system for cooperative problem solving is the group strategies: the meta-level knowledge for making the group of agents work with each other effectively. (2) Studying the coordination strategies for cooperative problem solving. For each individual problem-solving node(synonymously, agent),

the challenge is to coordinate in a most efficient and goal optimizing way its interactions with other agents. Coordination can be exerted in the form of passing data, goals, preference, partial solutions/plans, or constraints. (3) Constructing a framework for modeling group intelligence. Everyone agrees that " two heads are better than one." There are something about having others to work with in a problem-solving situation - a group always seems to be capable of achieving more than the sum of what the individuals can achieve. We call this phenomenon group intelligence. The important issue there is how group intelligence is formed and how the process can be modelled. This paper addresses these issue and illustrates the implementation of the cooperative problem-solving framework in an experimental network of expert systems - the Networked Expert System Testbed (NEST). The multi-agent problem solving system discussed in this paper is an information system consisting of a network of intelligent nodes capable of performing problem solving, referred to as agents. An example of such a system is a network of rule-based expert systems, such as NEST, each of which incorporating an area of expertise as represented by the content of its rules.

After reviewing the research issues related to multi-agent problem solving and developing a framework for characterizing such systems in Section 2, Section 3 focuses on the coordination mechanisms in multi-agent problem solving, which have direct bearing on the problem-solving strategies and learning schemes used. Section 4 describes various types of computer-based systems for cooperative work and group decision support. In Section 5, the possible schemes for carrying out learning in multi-agent problem solving are discussed. Section 6 illustrates the implementation of the framework in a network of expert systems - the Networked Expert System Testbed - for group problem solving.

2. A Framework for Multi-Agent Problem Solving

2.1 A Taxonomy

The multi-agent problem solving system discussed in this paper is an information system consisting of a network of intelligent nodes capable of performing problem solving, referred to as *agents*. An example of such a system is a network of rule-based expert systems, each incorporating an area of expertise as represented by the content of its rules. When such a multi-agent system is presented with a problem to be solved by the group of agents, the system needs to figure out a strategy so that the problem can be solved in the most efficient way by the group collectively. The intellectual leap from developing artificial intelligence for modeling human problem solving to developing DAI for multi-agent problem solving leads to a whole new uncharted area for research. The considerations here in many ways are similar to those involved in assigning a group of people, with varying areas of specialties, to solve a set of problems.

There are three different types of multi-agent problem-solving systems that have been developed:

Type 1: **Distributed problem-solving systems.** In these systems, the overall problem to be solved is decomposed into sub-problems assigned to the agents, each agent, asynchronously, would plan its own actions and turn in its solutions to be synthesized with the solutions of other agents. The agents in these systems use either *task sharing*(e.g.,Smith[1980]) or *data sharing*(e.g., Lesser&Corkill[1981]) to cooperate with other agents. The office information system described in Woo&Lachovsky[1986] and the scheduling system describe in Shaw&Whinston[1988, 1989] are examples of these systems.

Type 2: **Collaborative reasoning systems.** The agents in this second type of systems would be solving the same problem co'laboratively. The main issue here is not the decomposition into sub-problems assigned to the agents, but the solicitation of contributions from the participating agents so that the problem can be solved jointly by the group, simultaneously. The PLEXYS system(Nunamaker et al.[1988]), COLAB(Stefik et al.[1987]), and the concurrent design system described in Bond[1989] all fall into this category

Type 3: **Connectionist systems.** The third type of multi-agent systems use agents as the basic computational elements. These agents individually are just simple computing units and they not intelligent; but together they can solve complicated problems quickly. A typical example is the society-of-mind model described in Minsky[1985]. Unlike the previous two types of systems, where the agents are intelligent problem solvers, the agents in the connectionist model are only simple computation units. As a resul', they are restricted to perform simple tasks. The agents learn to solve problems more effectively by adjusting their connections with each other.

2.2 The Strategic Factors in Multi-Agent Problem Solving

Although the agents, the problems to be solved, and the strategies used are different in the afore-mentioned systems, it is nevertheless possible to constr.:t a general framework to describe the strategic factors shared by these systems. These factors are summarized below.

(1) **Goal Identification and task assignment.** There are two different types of problem-solving processes that can be used by the group of agents: in the first type, the problem is presented to the whole group and the agents would collectively carry out the deliberation process, which usually consists of issues identification, proposing solutions, discussion, prioritizing, and finalizing the group solutions. For the second type of group problem solving, the tasks involved in having the problem solved are structured and known, and the common strategy used consists of four steps. problem decomposition, task assignment, local problem solving, and solution synthesis.

(2) **Distribution of knowledge.** In designing a distributed knowledge-based system, the set of domain knowledge possessed by the group of agents can be distributed in different fashions.

In a sense, the consideration of knowledge distribution is similar to the design consideration of file placement in distributed databases - the bodies of knowledge should be placed according to the utilization demand of them by each agent. There may be different degree of redundancy in the agents' knowledge bases. In one extreme, the group of agents have exactly the same knowledge; in the other extreme, each agent in the group possesses different area of knowledge, without any overlap whatsoever. For some systems, however, the knowledge distribution is given and fixed.

(3) **Organization of the agents.** The control structure is determined by the organization of the agents. The group, for example, can have a central coordinator which gives commands to the other agents in a hierarchical fashion. The agents alternatively can have a flat structure, sometimes with the flexibility of dynamically forming sub-teams. In addition, the agents may also use a market structure, such as the case for the DAI systems based the contract-net framework. The organizational structure incorporated in the multi-agent problem solving system would dictate how the group of agents coordinate(Malone[1987,1988]).

(4) **Coordination mechanisms.**Coordination is necessary in multi-agent problem solving for resolving conflicts, allocating limited resources, reconciling different preferences, and searching in a global space for solutions based on local information. Coordination mechanisms can be based on a variety of information passed among the agents, such as data, new facts just generated, partial solutions/plans, preferences, and constraints. A number of coordination mechanisms have been developed for DAI systems. Each mechanism has its unique protocol for determining the timing of activities, triggering of events, action sequences, and message content in the coordination process. The role of coordination in multi-agent problem solving is discussed in detail in Section 3.

(5) **Learning schemes.** Learning should be an integral part of problem solving for improving the strategies, knowledge, and skills employed in the process. There are several learning processes that can be incorporated in multi-agent problem solving systems. It can be in the form of data exchange, knowledge transfer, or heuristic migration, where the learning mechanisms involved are relatively simple. It can also be done by extending machine learning techniques to single-agent systems, such as explanation-based learning, case-based reasoning, or inductive learning, to the multi-agent systems, where one agent can learn by observing other agents. Of particular interests are the use of group processes often used in human decision-making situations, such as group induction, nominal group techniques, or brain storming, for achieving the learning effects among the whole group. These group processes use structured sessions of information exchange to develop new concepts, which would lead to solutions not attainable by any of the agents alone. They are also good examples for illustrating the complementary role played by the problem-solving and learning activities.

2.3 Group Meta-Knowledge

In implementing a multi-agent problem-solving system, the afore-mentioned five strategic factors can be implemented in the form of group meta-knowledge. The group meta-knowledge would contain a variety of group functions and strategies for the group to conduct problem solving effectively. An example of such implementation is described in Section 6. Directiv affecting how the multi-agent system operates, this group meta-knowledge can be embedded in the network through which the agents communicate. Smith[1980] described the use of a 'problem-solving layer' for incorporating such group meta-knowledge in the layered network architecture. It can also be viewed as a 'protocol' which every agent has to observe for the group to operate efficiently. Depending on the organization of the system and the knowledge distribution, group meta-knowledge would be stored in the agents' knowledge bases. It is used by the multi-agent system to direct and coordinate the group problem-solving sessions.

3. The Role of Coordination

Coordination is the key design component for multi-agent problem-solving systems. Since each problem-solving agent only possesses local view and incomplete information, it must coordinate with other agents to achieve globally coherent and efficient solutions. The design of coordination can be viewed from three different perspectives: the information content, the exercise of control, and the coordination mechanisms. Coordination can be achieved through passing different types of information among the agents, such as data, new facts just generated, partial solutions/plans, preferences, and constraints. The initiative to coordinate may result from a variety of means of control: it may be self-directed, externally directed, mutually directed, or a combination of them(Lesser&Corkill, 1984).

Since coordination represents the major design components in extending problem-solving methods to the multi-agent environment, it has attracted the most attention for research in the literature. Many coordination mechanisms have been developed for DAI and multi-agent problem solving systems. Some more representative ones are described as follows:

1. **Coordination by revising actions.** One of the primary factor that necessitates coordination is the potential conflicts underlying the actions of the individual problem-solving agents. The objective of coordination, then, is to derive a solution or a plan of actions for the group such that all the conflicts are avoided. Cammarata et al.[1983] used the collision avoidance problem in air traffic control to illustrate coordination mechanisms based on passing the information constantly among problem solvers, each of which guides the course of an aircraft, and revise the flight plan if there is a danger of conflict with the plan of aircraft. Baskin et al.[1989] developed a conflict-resolution framework for coordinating the solution processes among multiple agents performing design tasks.

2. **Coordination by synchronization.** In a multi-agent system, the action of an agent usually affects some other agents. The objective of coordination in a way is to regulate the interactions among agents and to control the timing and sequence of these interactions. Corkill[1979] and Georgeff[1983] incorporated communication primitives, similar to the ones used in distributed operating systems, for synchronizing the problem-solving process of the agents.

3. **Coordination by negotiation.** Negotiation is a widely used form for mutually directed coordination: the process involves two-way communication to reach a mutually agreed course of actions. The contract-net mechanism described in Smith[1980] uses negotiation to coordinate the sharing of problem-solving tasks among agents, in which the negotiation process takes the form of contract bidding. Croft&Lefkowitz[1988] adopted the negotiation process to maintain the consistency of different agents' plans. Cammarata et al. [1983] also used the negotiation for finalizing revision of the agents' plans. Sathi&Fox[1989] develped a contraint-directed negotiation approach for coordinating the agents in resource reallocation.

4. **Coordination by structured group mediation.** Structured group processes, such as the nominal group technique, the Delphi technique, and the brainstorming process, are a special type of coordination based on multiple rounds of interactions with fixed sequence of group interactions. They are developed for guiding the group to reach satisfactory solutions in a systematic manner. Recent work on collaborative work (Stefik[1987]) and group decision support (Nunamaker et al.[1988]) use these group processes to mediate group problem solving among multiple human agents. It should be feasible, but yet to be tested, to implement these schemes for coordinating the problem-soiving activities among AI agents.

5. **Coordination by opportunistic goal satisfaction.** The blackboard model for problem solving(Nii et al.[1989]) has been used extensively in the multi-agent environment (Lesser&Corkill[1981], Durfee et al.[1985, 1987]). The blackboard model provides a paradigm for coordinating the multiple problem-solving agents(i.e., knowledge sources), who share the blackboard, to opportunistically contribute to the group solution process. However, the use of a single blackboard in a DAI system implies shared memory. Herman[1989] and Harrow [1989] incorporated a distributed version of the blackboard model but retain the opportunistic mechanism for coordinating the agents, each of which is a stand-alone expert system. Nii et al.[1989] described two DAI systems, Cage and Poligon, that extends the blackboard model to concurrent problem solving.

6. **Coordination by exchanging preferences.** A school of researchers have taken the game-theoretic view to mu ti-agent problem solving. They focus on how should the group of autonomous, self-interested AI agents interact with each other in achieving globally satisfactory

supports simultaneous activities among its agents, which are knowledge sources (KS) consisting of a human expert, a machine-based expert system, a software program, or any combination of these. Each knowledge source has a different area of expertise. The CPS system considers problem decomposition, solution synthesis, and planning of communication among the knowledge sources.

The KSs communicate by modifying a partial solution stored on a global blackboard. *Cooperation* is achieved by consulting constraints placed on the blackboard by other KSs, and by creating new constraints if the current ones are insufficient to produce a solution. These new constraints are then explored until a solution is found or they are proven incorrect.

Bond[1989] described a distributed knowledge-based system for concurrent engineering. A model for collaboration was constructed for coordinating communication activities such as suggestions, evaluations, and elaboration. Coordination strategies were used for the design environment.

5. Learning Schemes in Multi-Agent Problem Solving

5.1 Group Learning by Induction

Machine Learning is the study of how to develop computer programs that enables an intelligent system to learn. By learning we mean the system can improve its performance through acquiring new knowledge, refining existing knowledge, using better strategies, or memorizing cases previously proven to be successful. The whole research area of machine learning has produced fruitful results in the past decade and increasingly researchers have pointed out that learning capabilities should be an integral part of an intelligent problem solver.

In a multi-agent problem solving system, because of the interactions among the agents and the need for addition coordination, incorporating machine learning in the system become even more complicated than in a single-agent problem solver. However, potentially there are a number of benefits for developing machine learning techniques for multi-agent systems. First, multi-agent systems provide parallel processing capability in performing the learning the process, thus can achieve substantial speedup in learning new knowledge. Second, machine leaning capabilities can enhance the problem-solving ability of the multi-agent system and improve its performance. Third, understanding the learning processes in a multi-agent system can help develop better problem-solving strategies and coordination mechanisms. *It is our view that a group collectively can perform problem-solving tasks better than what the sum of the individuals' abilities primarily because of the learning processes that constantly go on in group problem solving.* Therefore, it is important to understand what the learning processes in group problem solving are.

Before we tackle the problem of group learning, let us first review the general computational model for achieving machine learning. Concept learning is one of the most developed areas in machine learning, where the objective is to derive concept descriptions satisfying all the training

examples. Starting with the given set of training examples, the learning process employs a series of generalization and specialization steps to search through the space of all possible concept descriptions; this process continues until the concept descriptions that satisfy all the descriptions of the training examples are found(Shaw[1987]). The whole process usually consists of several main steps:

Algorithm Concept-learning;
Input: a set of training examples;
a set of generalization and specialization rules;
Output: concept descriptions satisfying all the training examples;
Begin
(1) read in the next training example;
(2) generating new hypotheses about the possible concept descriptions based on this training example;
(3) evaluating the hypotheses by matching them with observations from the training set;
(4) based on the evaluation, adjust the hypothesis about the concept descriptions;
(5) if the new hypothesis, together with previous hypotheses generated, satisfy all the remaining training examples, stop; otherwise, go to (1);
End

In any machine learning program, there are two major components of the program that are crucial to the success of the learning performance. First, the proper mechanism for evaluating the hypotheses. This is called the *credit assignment* function. Second, the proper mechanism for generating new hypotheses based on existing ones. This is called the *hypotheses transformation* function. These two functions are key to all the machine learning programs developed. In the Concept-Learning Algorithm described above, step(3) is for credit assignment and steps (2) and (4) are for hypotheses transformation. Since this type of programs attempt to derive descriptions that can explain the given set of observations, or examples, the method is sometimes referred to as *inductive learning.*

To extend such a learning procedure to the multi-agent environment, additional *coordination* would be needed so that both credit-assignment and hypotheses-transformation functions can be performed in a global fashion. The idea here, however, is different from developing a parallel processing version of the learning algorithm to be performed by computer with multi-processors, in which case the computation-time reduction is inversely proportional to the number of the processors. More importantly, we want to emulate the group learning behavior which, through the process of knowledge fusion, can achieve more than the sum of the problem solvers involved. This is the essence of what we refer to as group intelligence.

A direct extension of the concept-learning algorithm for single agent is the *group induction* process. In group induction, the agents of the system engage in collecting evidence, generating hypotheses, and evaluating hypotheses in order to discover new concepts collectively. Each

support collaborative processes in face-to-face meetings. The objectives were to make meetings more effective and to investigate how computer tools affect meeting processes. In their design, Stefik et al. articulated the fundamental requirement for the system to provide a coordinated interface for all participants. With shared information presented to all the participants, they achieved the effect of *WYSIWIS* (what you see is what I see).

COLAB incorporated a number of tools(i.e., groupware) for group problem solving. For example, Cognoter is a COLAB tool for preparing presentations by a group. Cognoter organizes a meeting into three distinct phases - brain storming, organizing, and evaluating. This process can be extendible to the general group planning situations. Another example of the tool is Argnoter, a groupware for presenting and evaluating proposals. The Argnoter meeting comprises three distinct phases - proposing, arguing, and evaluating. The fact that these group processes for various functions are so similar point to the possibility of a designing a generic group problem-solving process. We present such a scheme in Section 5.

Cooperative GDSS

The Cooperative Group Decision Support System (Bui and Jarke 1984) has a distributed architecture of individual DSS connected to a group DSS via an interface component. The user only interacts with the individual DSS; at no time does the user have direct interaction with the GDSS. This system was developed for a decision situation with these characteristics: there are multiple decision makers with an equal weight in the decision making process; they interact cooperatively; the group has the same set of feasible decision alternatives, subject to a ranking according to specified criteria; and each decision maker has objectives that can be expressed by measurable criteria.

The individual DSS provides decision support for the individual and negotiation advisory support for the individual to interact with the other members of the group. The group DSS automatically selects the appropriate group decision technique, computes and explains the group decision, and suggests other ways to discuss the problem if a consensus cannot be reached. The DSS consists of content-oriented models (simulation, time series), a multiple criteria decision model, and group process-oriented models (Delphi, Nominal Group Technique, free discussion).

Participant Systems

A Participant System, as defined by Chang (1987), is a "computer system that facilitates the simultaneous interaction of several persons or intelligent agents working together...on a shared complex task." A participant system operates in real-time so that traditional human communication patterns (i.e. action and response) will not be lost. Communication occurs at the task level and metatask level to coordinate the group's interactions. Users receive different views of the problem representation, depending upon their role, as well as sharing common action,

visual, and cognitive spaces.

Participant Systems can be totally centralized, mixed, or totally distributed. Chang suggests that a mixed system could be developed by extending expert systems so that they can be used by several users, each with a different viewpoint, to collaboratively solve a problem.

Distributed Office Information System

Work in the area of distributed office problem solving, although not exactly the same as group decision making, addresses many of the issues encountered in developing our model. The following characteristics of office work, delineated by Woo and Lochovsky (1986), are applicable to collaborative problem solving: 1)Individuals must cooperate in order to accomplish many tasks, since one person cannot know how everything is done. 2)Work is performed concurrently. 3)Office workers derive their own rules and procedures for accomplishing a task. Thus, two workers may do the same task differently. 4)Workers use inputs and produce outputs. The inputs may come from another office worker; the outputs may be used by a co-worker. 5)Some information that is necessary to complete a task may be missing. The worker then has to ask another worker for help. Therefore, direct communication is essential. 6)Office workers know how their specific task fits into the overall task. However, they do not need to know about the goals of other workers unless it is necessary for their own task.

In their model, office knowledge is stored in independent knowledge bases at local sites. When a consensus is needed, they can cooperate. However, knowing where to go for help presents another problem since this is a distributed environment. A locator module is responsible for information travelling, communicating, and bringing back the information during a problem solving session.

Croft&Lefkowitz[1988] described a distributed knowledge-based system for supporting cooperative activities in the office environment. The main decision-support designs are incorporated in the forms of maintaining a representation of the environment that is shared by the problem solvers or agents, assisting in the execution of activities by formulating and executing plans, initiating negotiation between agents when actions cause plan failure, and assisting in negotiation initiated by agents.

Concurrent Engineering Design Systems

Parallel to the evolution of information systems in the office environment, the information systems for engineering design also have undergone major changes. One of the major objectives for concurrent engineering design is to coordinate the interactions between different departments and engineers involved in the process to shorten the design life-cycle and increase the products' competitive edge. Distributed knowledge-based systems have been developed for this purpose(Baskin et al.[1989], Bond[1989]). The system described in Baskin et al.[1989], called CPS,

solutions. Rosenschein et al [1984], Genesereth et al.[1986] developed a coordination scheme that does not need communication. The agents, however, need to know each other's possible actions and the corresponding utility functions. Taking a game-theoretic approach, Rosenchein et al[1985] described a coordination scheme between agents based on the exchange of the payoff matrix.

4. Computer-Based Cooperative Work and Group Decision Support

4.1 Overview

An important application of the multi-agent problem solving system is the group decision-support systems(GDSS). The demand for GDSS arose due to two conflicting organizational situations: the requirement for more information sharing in organizations as a means to cope with the environmental complexity and dynamism, and the resistance to managers spending so much time on meetings(Huber, 1984). A GDSS helps to solve this conflict. Its purpose is to "increase the effectiveness of decision groups by facilitating the interactive sharing and use of information among group members and also between the group and the computer"(Huber, 1984).

A group decision support system improves the process of group decision making through the use of communication, computing and decision analysis techniques. By effectively coordinating the exchange of information, it provides new approaches for organizational decision making and thus facilitates the solving of unstructured problems by a group. DeSanctis and Gallupe (1987) outline three levels of GDSS. Each level alters the group communication process to a different degree and requires more sophisticated technology. *Level 1* only facilitates information exchange by removing common barriers (i.e. electronic messages, anonymous input), so there is a small degree of communication change. *Level 2* makes qualitative modeling tools available to the entire group, thus reducing uncertainty in the decision making process. A greater degree of change occurs than with Level 1. Automated group structuring techniques, such as the Delphi method and Nominal Group Technique, can be applied at Level 2. A *Level 3* GDSS controls the communication process by imposing communication patterns on the group. Rules are developed to control the pattern, timing and content of the information exchange. The rules, which are applied to the group meeting procedure, can be pre-set during development or selected by the group.

Many aspects of the design of the GDSS are determined by the situation in which it will be used. DeSanctis and Gallupe classify GDSS according to the group size and member proximity as the following four types of systems:

1.The Decision Room - a smaller group with face-to-face communication. A GDSS would create an electronic meeting room, with devices for transmitting information among members, displaying data, and receiving data. Most current GDSS can be classified into this category.
2.The Legislative Session - a larger group with face-to-face communication. The Decision

Room technology would be enhanced to include a facilitator who controls the proceedings. Member to member communication may be restricted according to a hierarchical ordering or other rules. Level 3 software would be useful for enforcing meeting protocol.
3.The Local Area Decision Network - a smaller group with members dispersed. Electronic communication mechanisms such as LAN's, long distance networks, and teleconferencing would be an integral part of this type of GDSS. It also should allow for non-simultaneous meetings.
4.The Computer-Mediated Conference - a larger group with members dispersed. As with the Local Area Decision Network, communications and non-simultaneous usage are key aspects of the GDSS. A more structured dialogue between members could be controlled by facilitator software, hierarchical features (some menus only available to certain members) and Level 3 software that provides order to the conference.

4.2 Example Systems

PLEXYS

PLEXYS Planning System is an "automated system to support complex, unstructured group decision process" (Nunamaker et al. 1988). It is used in a decision laboratory by top managers as they carry on their planning and decision-making process. Managers sit at a U-shaped table equipped with networked microcomputers. A large screen projection system allows individual work to be displayed to the group. Qualitative and quantitative planning models are available to individual and group users to: 1) retrieve data from both internal and external sources, 2) analyze the data using the models, 3) form criteria, issues and assumptions relating to their decision, and 4) connect the assumptions and decisions, and save them for future use. The Session Management system provides access to the models by helping the user select the appropriate ones for the current situation. An agenda is developed that describes the planning and decision models to be used during the session, when and how to use them, and when discussions should be conducted.

Two of the software tools in PLEXYS are *Electronic Brainstorming* (EBS) and *Stakeholder Identification and Assumption Analysis*. EBS is based on the manual technique commonly used by groups. However, the process is automated and anonymity is maintained. The Stakeholder tool provides another way to identify issues critical to the planning process. Stakeholders pertinent to the proposed plan and their assumptions are identified, and then they are rated in terms of importance to the plan and to the stakeholders.

COLAB

The COLAB project was developed in Xerox PARC for using computer-based system to

agent makes its individual observations, forms hypotheses based on the observations, and keeps searching for new evidence for modifying the hypotheses. As more observations are made, evidence may confirm or disconfirm hypotheses, thus strengthening or weakening the agents' beliefs about the hypotheses. This process continues until new concepts are derived that are supported by the observations and the agents' beliefs. The crucial design in such a group induction process is the *coordination mechanism* incorporated for the agents to interact with each other. Proper coordination can greatly accelerate the learning process by having the agents share their beliefs as well as the newly generated hypotheses that appear to be successful.

Since there have not been any work done on developing group induction algorithms in the machine learning literature, a possible source of inspiration is from the psychology literature where researchers have been conducting behavioral experiments to study group induction. For example, Laughlin&Shippy[1983] described an experiment on group induction to address the issue whether " a cooperative group is able to induce a general principle that none of the group members could have induced alone, or the group merely adopt an induction proposed by one or more of the group members."

The experiment conducted was to ask a group human subjects to identify the rule used in partitioning decks of bridge playing cards. The group induction procedure begin with a known positive example of the rule. The objective is to ask the group to induce the rule in as few trials as possible. A trial consists of the following five stages: First, each individual group member wrote down an hypothesis (proposed induction). Second, the group members discussed until they reach a consensus on a group hypothesis. Third, the group members discussed until they reached a consensus on a play of any of the cards, which was shown to the experimenter. Fourth, the experimenter classified this card as either a positive or a negative example. Fifth, the experimenter checked the group hypothesis. if the group hypothesis was correct the problem was solved; if the group hypothesis was incorrect, a new cycle of five stages followed, consisting of a second cycle of individual hypothesis, group hypothesis, experiment with a play of a card, feedback on the status of the card, and check of the group hypothesis as correct or incorrect. The process terminated when a correct group hypothesis was found. Laughlin&Shippy[1983] concluded that the group were remarkably successful in recognizing and adopting a correct hypothesis if the hypothesis was proposed by one or more members. In contrast, group induction in the strong sense that a correct group hypothesis was derived that none of the group members had proposed individually was extremely rare. Their findings can be summarized by the following propositions.

Proposition 1. *Group induction is advantageous over single-agent induction primarily due to the agents' sharing of hypothesis, not necessary because of that the group together can generate new hypothesis not generated by the individual members.*

Proposition 2. *In the group induction process, the sharing of hypothesis among the agents would result in the identification of correct group hypothesis in fewer iterations.*

The first proposition attempts to identify the key element of group induction; the second proposition attempts to explain the underlying reason for the first proposition. Together, based on the evidence presented by the behavioral experiment, these two propositions point out the importance of information sharing in a group induction process. Designing algorithms for achieving the same learning effects, therefore, requires the incorporation of effective coordination mechanisms that would facilitate information sharing.The basic group induction process can be represented algorithmically as follows:

Algorithm Group-Induction
Input: a set of training examples;
a group of problem-solving agents;
a set of generalization and specialization rules for each agent;
Output: learned hypothesis generated by the group;
Begin
(1) a new training example is presented to the group;
(2) for each agent, a new hypothesis is generated based on the current and previous examples;
(3) for each agent, the agent's hypothesis is broadcast to the whole group;
(4) all the hypothesis are evaluated and modified if necessary;
(5) if a correct hypothesis emerge, stop; otherwise, go to step (1).
End

Note that in this group induction algorithm, the learning process is carried out by the set of agents collectively. This algorithm has the flavor of a parallel algorithm in steps (2) and (3) when the operations are performed on the set of agents concurrently. The major benefit of having the group of agents is their sharing of the newly generated hypothesis in each iteration, which increase the probability that a correct hypothesis can be identified sooner. In this particular algorithm, the information sharing is achieved by a simple broadcasting mechanism. In a DAI system, since each AI nodes, a problem-solving agent, has a different knowledge base, the rules they use in generating new hypothesis would be different. Information sharing thus would provide greater probability to identify the correct hypothesis for concept descriptions sooner.

We can extend the afore-mentioned propositions and make the conjecture that group problem solving can find solution quicker because of information sharing. New facts, hypothesis, and partial solutions generated by one agent in its problem-solving are shared by other agents. These additional pieces of information shared among the agents would trigger the set of agent to new problem-solving and evidence-gathering activities which would not have been performed without the agents' interactions. In turn, these triggered activities would generate new facts, hypothesis, and evidence that are shared among the agents. Thus, the amount of information shared in the

agents' problem-solving efforts grows exponentially. And this gives the group an advant ge to reach a solution much sooner. On the other hand, the exponentially growing information would mean substantially more searching efforts in screening through all the information generated. A properly designed *coordination mechanism* can lead the group to evaluate all the facts, hypothesis, new evidence, and partial solutions more systematically. These views can be summarized by the following propositions.

Proposition 3. Group problem solving has the advantage over single-agent problem solving in that the hypothesis, facts, evidence, part olutions can be shared among the agents, triggering additional problem-solving and evid -gathering activities that would not have been performed without the group. Because of these additional problem-solving activities, the group collectively would have a better chance of reaching a solution sooner.

Proposition 4. The information sharing activities would result in an exponential growth in the amount of information generated by the group problem-solving process. In order to focus on the most relevant information, coordination mechanisms would be needed to organize the group interactions and to guide the agents to use all the information that have been generated.

Proposition 4 further articulates our view that **coordination** is the key to multi-agent problem-solving. The primary objective stated here is for focusing on the relevant information. It also can be used to constrain the search space engaged by the group. The Delphi method is an example for such a group problem-solving method incorporating a structured coordination mechanism. Linston&Turoff[1975] define the Delphi technique as " a method for structuring a group communication process so that the process is effective in allowing a group of individuals, as a whole, to deal with a complex problem." In the Delphi method there are four phases to the group's interaction process: 1. Exploration of the subject to be discussed by the group; 2. reach understanding of how the group views the issues; 3. if disagreements exist, explore reasons and evaluate; 4. when all information has been analyzed and evaluations have been fed back for consideration, make a final evaluation. To accomplish structured communication, the coordination mechanism underlying the Delphi technique needs to provide: some exchange of individual contributions of information and knowledge; some assessment of the group judgement and view; and some opportunity for individuals to revise views(Linston&Turoff[1975]). This four-stage Delphi procedure is very similar to the aforementioned group-induction process in that the emphasis of group interactions is placed on sharing information. This similarity also underlines the tight relationship between group problem solving and group learning.

In a DAI system, integrating group problem solving and learning is useful. In the application of using a distributed knowledge-based system for concurrent design, for instance, group problem solving can be used to integrate the various views taken by the problem-solving agents with different area of specialization, with a coordination mechanism similar to Delphi

incorporated to adjust the individual solutions. When there is a conflict, the conflicting views are explored and the underlying reasons for the conflict would be identified. This view deviation will be used as a basis to modify the knowledge bases. Such a process is basically a knowledge refinement procedure based on the group interactions, as opposed to be based on the discrepancy between the ideal solution and the actual solution as in a single-agent situation. After the agents' views have been adjusted, the subsequent problem-solving process for the group can be facilitated.

5.2 Group Learning by Genetic Adaptation

Shaw and Whinston[1989] describes a learning scheme for a group of agents based on the genetic algorithms and the learning classifier systems discussed in Holland[1975] and Holland et al.[1986]. A bidding mechanism is incorporated for credit assignment among the agents: the agents who have more contribution to the problem-solving process would get more credit, and the credit each agent accumulated in turn would be used as the fitness measure. The genetic adaptation plan is performed on the group of agents to refine the knowledge (that is, knowledge about solving the problem as well as group meta-knowledge) and its distribution.The transformation of agents' knowledge is achieved by such genetic operators as reproduction, crossover, and mutation. Shaw and Whinston[1989] applied the group learning technique to the scheduling and configuration/reconfiguration problem in flexible manufacturing systems.

Shaw[1990] described a distributed problem-solving approach to group induction, in which the learning problem is decomposed into n subproblem. Each subproblem is solved by an inductive learning algorithm(PLS1); the solutions to the subprobelms are then used as the input for the genetic adaptation procedure. Computational results showed that such a decompositional approach can improve the learning performance.

5.3 Learning in the Society of Mind

Minsky[1985] developed a model for intelligence based on the theory that a human problem-solving process consists of a set of smaller processes, each conducted by an *agent* with specialized responsibility. The problems handled by the agents in Minsky's model, referred to as the society of mind, have much smaller granularity than ones handled by the agents described in this paper. The major thrust of the society-of-the-mind model is to explain intelligence as a combination of simpler things executed by agents. Although none of these agents individually is intelligent, collectively they can solve complicated problems and do that quickly. This concept, presented in a more microscopic level, is very similar to our observation of group intelligence described in the preceding section.

The society of the mind models both the problem-solving as well as the learning processes The key is to put importance on not only how the agents solve the sub-problems, but also on the

inter-relationships between the agents in the whole process. The idea is that in the course of solving some problem, certain agents must have aroused certain other agents. A reward system is incorporated so that if agent A has been involved in arousing agent B, it will be easier for A to arouse B in the future. The group of agents activated in having a problem solved would have strong connections with each other. Such connected groups should make it easier to solve the same problem the next time by simply reactivated the same group. Minsky developed a theory of memory based on this connectionist view, in which the group of agents involved in solving a problem successfully was referred to as the *knowledge line*.

Learning, based on this model, is not simply the acquisition of a single concept. It involves assigning priorities(weights) to competing concepts which can be represented by a hierarchy of agents.A crucial aspect of the model is to deal with knowledge refinement in such hierarchies. Minsky argued that it is not enough to have many kinds of reasoning; one must know which to use in different circumstances - i.e., the knowledge about how best to use what was already learned. Minsky related this to the issue of the organization of the agents in the following characterization, what he referred to as the Papert's principle:

Papert's Principle: *Some of the most crucial steps in mental growth(i.e., learning) are based not simply on acquiring new skills, but on acquiring new administrative ways to use what one already knows.*

In other words, in the multi-agent system, it is necessary to learn how to **organize** the group of agents in a problem-solving process most efficiently for solving that problem. In the society-of-mind model, as the problem-solving process help accumulate more low-level agents, it is necessary to create new levels of agents to administer the low-level agents. The group of agents thus would grow increasingly into a multi-level hierarchy. This emphasis on organizational structures in dealing with the multiple agents is consistent with our framework described in Section 2. Since this model uses fine-grained problems, the agents are simple computation elements. Hierarchies can provide good coordination for the group of agents with the least amount of overall communications activities required(Simon[1982], Malone[1987,1988]).

6. Implementation: A Networked Expert System Testbed(NEST)

A prototype version of our multi-agent model, based on the framework discussed in Section 2, was implemented. The system, referred to as the networked expert systems testbed, or NEST, consists of a network of four expert systems. The architecture of NEST is based on a variation of the blackboard system, with mailbox areas added to the blackboard shared area for coordinating the agents. The architecture of NEST is shown in Figure 6.1.

Insert Figure 6.1 Here

NEST was developed using Texas Instrument Personal Consultant Plus Version 4.0, a frame based expert system development program. Personal Consultant On-line, a supplementary tool for real-time applications, was also utilized as needed. The prototype was installed on an IBM Token Ring Network of IBM-AT computers. PC Plus and On-line were installed locally at four terminals of the network. A distributed version of dBase III+ was installed as a shared memory so all nodes could have access to it. dBase was selected because PC Plus has internal interface commands with it.

6.1. Group Meta-Knowledge

The groupware expert system is generic. It was not developed dependent on a specific application. The **group meta-knowledge** of the system direct the group problem-solving processes. It asks the user for the problem description and characteristics of the problem; It also determines some group characteristics and selects the appropriate **group strategy** based on this information.

Corresponding to the **knowledge distribution**, the user describes the group problem with the aid of an *expertise* database. This contains a list of the available expertise on the network, indexed by a standardized classification scheme. It also indicates the node where the expertise is located. The database needs to be updated by a human system administrator whenever a new expert system is added to the network.

For the **goal-identification and the task-assignment** function, if the user is unable to describe the problem with the given expertise, the problems may be decomposed into sub-problems. For example, the goal is to decide sales for combined sales regions A, B, and C, but the expertise does not exist for the combined regions. Since expertise does exist for region A, region B, and region C separately, the user can create sub-problems for each region. The group knowledge base will select a strategy, etc. for each of the sub-problems individually. The user can then combines the results from each region and arrive at the overall problem solution. This is the *task-sharing* strategy for distributed problem solving.

The expertise database is also used for task assignment. Given the problem description, the expertise database is consulted to find out which nodes are capable of solving this problem. This is especially helpful in the collaboration strategy. When one node needs help and asks the strategy coordinator for assistance, the coordinator checks in the expertise database to see which nodes can provide the required value

6.2. Coordination Mechanisms

NEST uses a modification of the blackboard architecture, in which the agents, i.e., the expert

systems, can communicate through two means - the **blackboard** or the **mailbox** - for coordination."

Blackboard

The blackboard communication area is implemented with the following three databases:

(1)*Strategy database*

This serves as a mailbox for the strategy coordinator to coordinate agents. It is a part of the blackboard and all of the local nodes have access to it. The fields of the database indicate who sent the message, a command for the coordinator, and a message. The collaboration coordinator could receive two types of messages: 1) Node-n REQUEST QUANTITY - Node n is requesting the value for parameter QUANTITY; and 2) Node-m RESPOND DONE - Node m is informing the strategy coordinator that it has completed its task.

(2) *Value database*

Serving as the world model for the multi-agent environment in NEST, this is a shared area where parameter values for the current problem are stored. It contains the parameter name, the value, and a timestamp. Every time a node ascertains a new value for a parameter, it updates the entry in the value database. Thus, the most recent value of the parameter is always available. Users of the value database (i.e. other nodes) can calculate how current the value is by checking the timestamp. If the value is current enough, then the value can be retrieved from the database and further calculations are not necessary.

(3)*Status database*

This database marks which nodes are currently busy so they will not be assigned a new task. A timestamp is included for start and finish of the task.

Mailbox

The second means of communications between the agents are through the mailbox area. Each node in the network has a unique mailbox database that serves as its mailbox (refer to Figure 6.1). Messages to the node from the coordinator are placed in this database. The message consists of a field indicating the type of message (provide), and a field containing the parameter name. For example, the message PROVIDE QUANTITY is used to indicate that the node is requested to provide the value for the parameter QUANTITY. The strategy coordinator would initiate a functional expert system by sending it this type of message. The message database is periodically checked by each node to see if it contains any new messages. Just like the way one would check his mailbox.

State Transitions in the Expert System Nodes

There are three possible states that an expert system node can be in:

In **Idle state**, the node is waiting to be initiated by a strategy coordinator or by a human user. The node periodically checks its message database to see if it has any "provide" messages. If so, then it enters the *working state*.

The node is actively running a consultation during the **working state**. When information is needed, it checks the value database for the required parameter value. If the value is not there, or if it is too old, then the functional expert system sends a "request" message to the collaboration coordinator (Strategy Database) asking for help. It then enters the active *waiting state*. It returns to the *working state* when the parameter has a new value and the consultation continues until complete. At that time, the node will update the value database with new parameter values, and send a "respond" message to the strategy coordinator. The node re-enters the *Idle state*.

In the **active waiting** state, the node periodically checks the value database until the requested parameter value is updated. If possible, the node will continue to run the consultation, determining other parameter values and goals that do not rely upon the missing information.

6.3. Organization of the Expert Systems

The organization incorporated in NEST is a hierarchical structure, in which an expert system serves as the coordinator in the group problem-solving process. Given the initial problem description, the collaboration coordinator assigns the problem to a functional expert system node. It uses the expertise database to check the knowledge distribution and see which nodes are capable of solving this problem, and checks the status database for their availability. When the assignment is made, the functional expert system is initiated by sending it a "provide" message.

The coordinator periodically checks in the strategy database for messages from the functional expert system. In the collaboration strategy, two messages are possible. A local expert system will send a "request" message when it needs a parameter value. The coordinator determines which nodes can supply this parameter value, assigns the task to one of them, and updates the status database to reflect the assignment.

A "respond" message can also be sent to the strategy database. This indicates that a node has completed its processing. The coordinator updates the finish time in the status database. If all nodes in the status database are done, then the collaboration process has solved the group problem.

6.4. A Distributed Problem-Solving Example

To demonstrate the working of NEST, in particular the collaboration strategy, a common business situation was used. The knowledge distribution is as follows: Three functional areas - marketing, production, and purchasing - need to collaborate to reach a decision: What quantity of a product should be sent to the market? Perhaps this decision could influence the amount of

advertising and sales effort.

The marketing expert initially sets a desired market quantity, based on given supply and demand information. Before it can make the final decision, Market Quantity, it needs input from the other two experts: Production Quantity and Direct Labor Cost from Production; Direct Materials Cost from Purchasing(Figure 6.2).

Insert Figure 6.2 Here

The purchasing expert receives the request for information from marketing, along with marketing's desired market quantity. Using its production constraints and other scheduling requirements, it determines how much it is capable of producing for marketing (Production Quantity). However, this quantity is also dependent on the availability of materials from the purchasing department. If the materials are not available, then the quantity will be adjusted accordingly before notifying marketing.

The purchasing expert determines if it has enough materials in inventory, or if it has to place an order, for production to manufacture its recommended quantity. It also calculates the direct materials cost for marketing. This decision process continues until marketing reaches a final decision. The market quantity will be adjusted in response to production's capabilities and the cost to produce. For instance, if the costs are too high, then the quantity may be increased to achieve economies of scale.

The design details of NEST and the trace of rules used in the group problem-solving session for solving this example are described in Harrow[1989] and Herman[1989]. Shaw et al.[1991] relate the construction of NEST to a DAI framework for group problem solving.

The implementation of NEST demonstrates the viability of applying the multi-agent problem-solving framework described in Section 2. It is especially interesting to be able to integrate two widely used software to implement the expert system nodes and the distributed databases, incorporating a variation of the blackboard model. More work would be needed to make the group meta-knowledge more flexible in dealing with different types of problems, group strategies, learning schemes, organization structures, and knowledge distribution.

7. Conclusions.

Multi-agent problem-solving systems with DAI capabilities have emerged as a new area of research. In this paper, we have shown the importance of coordination and learning in such systems. We also showed the application of the DAI framework to such applications as group decision-support systems, office information systems, and engineering design systems. Having integrated various developments in DAI, group decision support, and cooperative decision making, we applied the multi-agent problem-solving framework described to the implementation of a network of expert systems, incorporating a variation of the blackboard model. We are currently working on the group learning schemes for multi-agent problem solving and the supporting coordination mechanisms.

References

Baskin, A.B., Lu, S.C-Y., Stepp, R.E., and Klein, M., "Integrated Design as a Cooperative Problem Solving Activity," *Proc. Hawaii International Conference of Systems Science*, 1989, p. 387-396.

Bond, A.H., "The cooperation of experts in engineering design", *Distributed Artificial Intelligence*, vol. II., L. Gasser and M. Huhns (Eds.), Pitman, London, 1989, pp. 463-486.

Bui, T. and Jarke, M., "A DSS for Cooperative Multiple Criteria Group Decision Making," in Proceedings of the 6th International Conference on Information Systems (Tucson, Ariz, Nov.), 1984, pp. 101-113.

Cammarata, S. McArthur, D. and Steeb, R., "Strategies of cooperation in distributed problem solving", *Proc. 8th Int. Joint Conf. Artificial Intelligence*, 1983, pp. 767-770.

Chang, E., "Participant Systems for Cooperative Work," in Distributed Artificial Intelligence, M.N. Huhns (ed.), Pitman, London, 1987.

Corkill, D.,"Hierarchial planning in a distributed environment," in *Proc. Sixth Int: Joint Conf. on Artificial Intelligence*, Tokyo, Japan, Aug. 1979, pp. 168-175.

Croft, W.B. and Lefkowitz, "Knowledge-based support of cooperative activities", *Proc. 21st Annual Hawaii International Conference on System Sciences*, vol. III, pp. 312-318, 1988.

Dennis, A.R., George, J.F., Jessup, L.M., Nunamaker, J.F., and Vogel, D.R., "Information Technology to Support Electronic Meetings," MIS Quarterly, Dec 1988, pp. 591-624.

DeSanctis, G. and Gallupe, R.B., "A Foundation for the Study of Group Decision Support Systems," Management Sciences, May 1987, pp. 589-609.

Durfee, E.H., Lesser, V.R. and Corkill, D.D., "Cooperation through communication in a distributed problem solving network", in *Distributed Artificial Intelligence*, M. Huhns (Ed.), Pitman, London, U.K., 1987, pp. 29-58.

Genersereth, M.R., Ginsberg, M.L. and Rosenschein, J.S., "Cooperation without communication", *Proc. 5th National Conference on Artificial Intelligence*, Philadelphia, PA, 1986, pp. 51-57.

Georgeff, M., "Communication and interaction in multi-agent planning," *Proceedings AAAI-83*, 1983.

Harrow, B., *A networked expert system for group problem solving*, unpublished M.S. Thesis, Department of Computer Science, Univ. of Illinois, Champaign, IL, 1989.

Hayes-Roth, B., "A blackboard architecture for control," *Artificial Intelligence Journal*, vol. 26, pp. 251-321, 1985.

Herman, S., *Group problem solving with distributed expert systems*, unpublished M.S. thesis, Department of Computer Science, Univ. of Illinois, Champaign, IL, 1989.

Holland, J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

Holland, J.H., *Induction: Processes of Inference, Learning and Discovery*, MIT Press, Cambridge, MA, 1986.

Huber, G.P., "Issues in the design of group decision support systems", *MIS Quarterly*, 1984, vol. 8, no. 3, pp. 195-204.

Jarvenpaa, S L., Rao, V.S., and Huber, G.P., "Computer Support for Meetings of Groups Working on Unstructured Problems. A Field Experiment," *MIS Quarterly*, Dec 1988, pp. 645-666.

Laughlin, P.R and Shippy, T.A., "Collective Induction," *J. of Personality and Social Psychology* (45.1), 1983, pp. 94-100.

Lesser, V.R. and Corkill, D D , "Functionally accurate, cooperative distributed systems," *IEEE Trans. on Systems, Man, and Cybernetics*, 1981, vol. 11, no. 1, pp. 81-96.

Linstone, H.A. and Turoff, M. (eds.), *The Delphi Method: Techniques and Applications*, Addison-Wesley, Reading, MA, 1975.

Malone, T.W., "Modeling coordination in organizations and markets," *Management Science*, 1987, vol. 33, pp. 1317-1332.

Malone, T.W. and Smith, S.A., "Modeling the performance of organizational structures," *Operations Research*, 1988, vol. 36, pp. 421-436.

Minsky, M., *The Society of Mind*, Simon and Schuster, Inc., N.Y., 1985.

Nii, H.P , Aiello, N., and Rice, J., "Experiments on Cage and Poligon: Measuring the Performance of Parallel Blackboard Systems," *Distributed Artificial Intelligence*, vol. II., L. Gasser and M. Huhns (Eds.), Pitman, London, 1989, pp. 319-384.

Nunamaker, J.F , Applegate, L.M. and Konsyuski, B.R., "Computer-aided deliberation: Model management and group decision support," *Operation Research*, 1988, vol. 36, no. 6, pp. 826-847.

Rosenschein, J S. and Genersereth, M.R., "Communication and cooperation, Technical Report HPP-84-5, Computer Science Department," Stanford University, Palo Alto, CA, 1984.

Rosenschein, J.S. and Genersereth, M R., "Deals among rational agents," in *Proc. 9th Int. Joint Conf. Artificial Intelligence*, 1985, pp. 91-99.

Sathi, A. and Fox, M., "Constraint-Directed Negotiation of Resource Reallocations," *Distributed Artificial Intelligence*, vol II., L. Gasser and M. Huhns (Eds), Pitman, London, 1989, pp. 163-194.

Shaw, M , "Applying inductive learning to enhance knowledge-based expert systems," *Decision Support Systems*, 1987b, vol. 3, pp. 319-322.

Shaw, M. and Whinston, A.B., "A distributed knowledge-based approach to flexible automation: The contract-net framework," *International Journal of Flexible Manufacturing Systems*, 1988, pp. 85-104.

Shaw, M. and Whinston, A.B., "Learning and adaptation in distributed artificial intelligence systems," in *Distributed Artificial Intelligence*, 1989, vol. II, L. Gasser and M. Huhns (eds), Pitman, London, pp. 413-430.

Shaw, M., Harrow, B. and Herman, S., " Distributed Artificial Intelligence for Multi-agent Problem Solving and Group Learning," forthcoming in *Proc. Hawaii International Con. of System Science*, IEEE Press, 1991.

Shaw, M. " Mechanisms for Cooperative Problem Solving and Multi-Agent Learning in Distributed Artificial Intelligence Systems," submitted to the 10th AAAI International Workshop on Distributed Artificial Intelligence, Texas, Oct. 1990

Simon, H A., *The Sciences of the Artificial*, MIT Press, Cambridge, Massachusetts, 1982.

Smith, R.G., "The contract net protocol: High level communication and control in a distributed problem solver," *IEEE Transactions on Computer*, 1980, vol. 29, pp. 1104-1113.

Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., and Suchman, L., "Beyond the chalkboard: Computer support for collaboration and problem solving in meetings," *Comm. Ass. Comput. Mach.*, 1987, vol. 30, no. 1, pp. 32-47.

Woo, C.C. and Lochovsky, F.H., "Supporting distributed office problem solving in organizations,"*ACM Trans. on Office Information Systems*, July 1986, pp. 185-204.

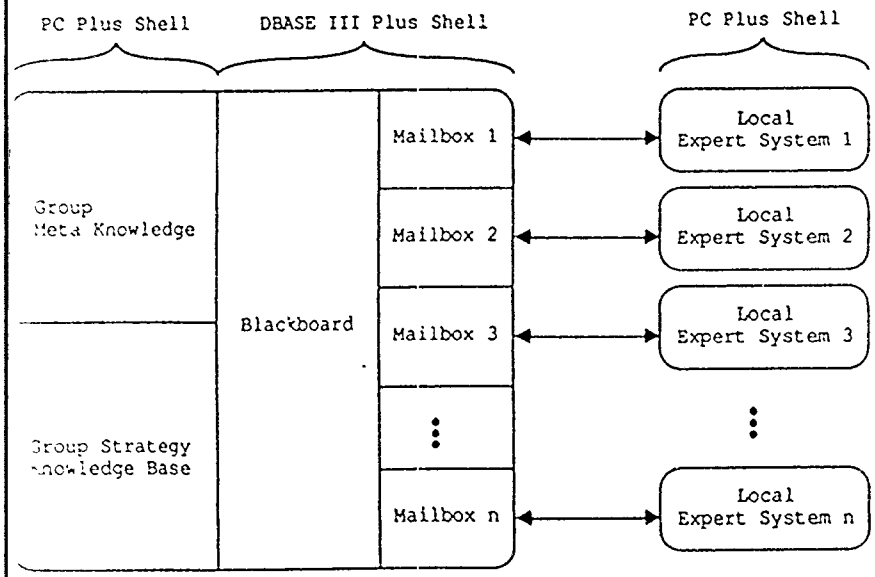


Figure 6.1 The Architecture of NEST

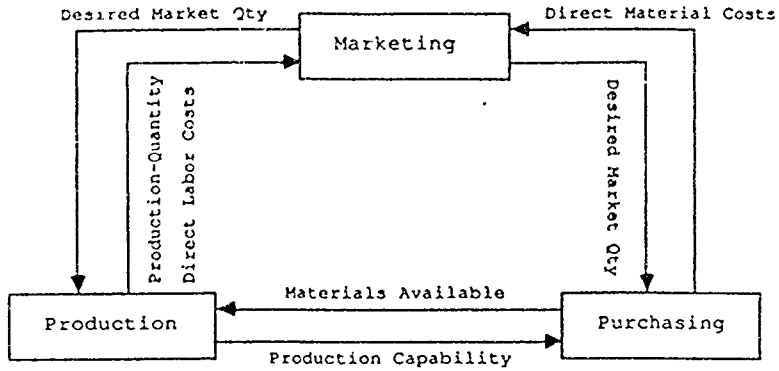


Figure 6.2 Knowledge Distribution and the Information Flows for the Example

Collaboration of Knowledge Bases via Knowledge Based Coordination

Donald D. Steiner
Siemens AG, DFKI
steiner@informatik.uni-kl.de

Dirk E. Mahling
DFKI, Univ. of Massachusetts
mahling@informatik.uni-kl.de

KIK Project Group
DFKI (German Research Centre for AI)
Postfach 2080, D-675 Kaiserslautern, Germany

Draft Paper for
International Working Conference on Cooperating Knowledge Based Systems
October 3-5, 1990

Abstract

To take advantage of the growing number of databases and knowledge bases, we present a system that allows for cooperation among spatially distributed agents Cooperation is supported by providing a uniform agent model and a framework for instantiating various cooperation strategies The knowledge base functionality forms an agent's body, while the cooperation functionality is achieved via the agent's head, which is a highly specialized coordination and communication expert system. The power of the cooperation arises from the ability to combine highly specialized knowledge bases and powerful reasoning engines to "expert system task forces". These task forces are created only for the duration of a certain project and modify themselves as the need arises. This flexibility brings the benefits of coordination technology from the area of computer supported cooperative work to distributed knowledge bases.

1 Introduction

As the tasks in our society grow in complexity, ways must be found to aid in the solution of problems in work and private life. Knowledge bases were developed to make a first step towards the solution of this problem. As is apparant now, isolated knowledge bases can cope with the demands of real world applications only in very special cases. It is therefore necessary to pool the knowledge resources available for a problem at hand. This means allowing knowledge bases to cooperate with each other and also to cooperate with humans. An example of such a situation is a knowledge base which needs to process knowledge beyond its capabilities and therefore must access another knowledge source. The major problem facing us then, is to allow independent, knowledge based agents (humans or AI systems) to coordinate their efforts. We present a model of cooperating agents, which works effectively for supporting cooperation among knowledge based systems. We view *cooperation* among agents as a two-step process. In order to effectively *collaborate* (work together in order to to solve a task), agents must first *coordinate* their task solving approach. This need not necessarily be done by all agents together (resulting in the cooperative task of deciding on the correct coordination), but may be handled by specialized coordination agents. Cooperation involves many iterations of coordination/collaboration. In this paper we discuss the efforts of Project KIK, concerning the cooperation among knowledge bases taking both steps into account. We also take advantage of the fact that knowledge bases can exist independently from inference engines.

Project KIK [1] brings Distributed Artificial Intelligence, Socio-Cognitive Engineering, Telecommunication, and Interface Design together in an attempt to provide integrated solutions to collaboration problems. Viewing human and machine agents as partners in communication, cooperation, problem solving, and task execution extends the conventional CSCW paradigm to Human-Computer Cooperative Work (HCCW).

The combination of rapid advances in CSCW, Socio-Cognitive Engineering, and DAI, as well as in network and communication technology, have the synergetic potential to design systems allowing for novel scenarios which substantially exceed the performance of current cooperation systems. We are developing a design paradigm for cooperative systems that starts by mapping the social, cognitive, and functional territory [Mah90]. This mapping process includes traditional task and goal analysis, ethnographic methods, and other behavioral and analytical techniques. After the mapping survey of users, tasks, goals, requirements, etc. has reached a stable level, the formation of a model can start. For this purpose, ex-

[1] Project KIK is a collaborative effort between the German Research Center for Artificial Intelligence (DFKI) and Siemens AG, the acronym KIK arises from the german abbreviations for *Künstliche Intelligenz (KI)*, meaning AI, and *Kommunikation*, meaning communications

isting models and theories in the scientific body of knowledge that can be used to embed, explain, or predict findings from the mapping process are indexed. Facts and observations that can not be covered in this way, possibly lead to additional theoretical research. Eventually a model will be synthesized that integrates data from the mapping process, existing models and theories, which are applicable, and sub models from additional research. Based on this synthesized model, implications can be drawn on how to design or modify the collaboration among human and machine agents. Rapid prototyping in conjunction with user-centered design and user participation s used to build initial versions of a solution to work and cooperation problems. These solutions are iteratively refined, until operational usability goals are reached [WBH87].

The goal of the TEAMWARE subproject of KIK is the specification and design of an architecture for a wide class of cooperation systems. In order to support the logical aspects of distribution and cooperation in such systems we are building the extended multi-agent framework MECCA (Multi-Agent Environment for Constructing Cooperative Applications). This framework, based on the general agent model, will be used as the underlying architecture for constructing CSCW applications.

In this paper, we describe the particularly important roles that knowledge bases play in the CSCW effort. We further present a general model of agents in distributed systems and demonstrate how this model will support the incorporation of knowledge bases into CSCW systems. We end with a particularly relevant CSCW application, namely that of constructing large knowledge based systems in a team.

Much of the research done in CSCW has focused on providing low-level support for communication between agents. It is our hypothesis that, in order to provide effective support and coordination, the system must contain a model of the overall environment and of the agents. The particular approach to knowledge based support that we are proposing emphasizes the use of knowledge bases, plans, and agent models. A plan-based approach has significant advantages in making knowledge based support feasible in environments which are dynamic and inherently open-ended.

2 Agent Model

In this section we describe a general model of an agent which was developed in order to support the wide variety of agents, communication structures, and cooperation structures required by the CSCW and DAI scenarios. The two major roles of such an agent are.

1. Execution of assigned tasks bringing the group closer to the top-level goal;

2 Participation in the cooperation process.

Execution refers to the agent's individual task functionality for performing subtasks in the application domain; the complexity of the individual skills of various agents may differ drastically (from, say, printing a document to highly sophisticated knowledge based deduction).

Participation refers to the agent's capability for performing cooperation tasks The central issue from a cooperative point of view is the additional amount of processing an agent must perform in addition to subtask execution in the application domain. This additional behavior of agents is based upon knowledge which explicitly represents the underlying cooperation model. This will generally involve interacting with other agents requesting information, giving information etc

Further, in order to cooperate effectively, an agent must be able to communicate with other agents in the system (i.e. send and receive messages to and from other agents) and to react appropriately upon receipt of a message (i.e. based upon the content of the message perform some task).

2.1 General Agents

An agent can be decomposed into the following parts [Con89]: the functional, task-solving component *agent body*, the cooperative superstrate *agent head*, the communication functionality *mouth*, and the physical connections to other agents via the communication channels.

The body of an agent constitutes its internal problem-solving expertise and provides the basic functionality of the agent: that portion which the agent can do independent of cooperation and on its own. An agent body can be a piece of hardware (e.g. sensor or robot) or software (e.g. financial advisor package) or humans with their respective skills. An agent body can be pre-existing hardware or software. It may be independent of any multi-agent system, as a stand-alone application, or participate in different systems. Once created, an agent is expected to follow the general rules of that multi-agent environment until released. Reusability and duplication of bodies of machine agents is generally possible. The head of an agent is that portion which allows the agent to participate in the cooperation process. For machine agents, it is generally a software system with several components. For human agents it will be comprised of the human's own knowledge and an intelligent user interface between the human and the other agents.

In order for an agent to contribute to the overall problem solving process in a goal oriented fashion it must have knowledge about:

- its own facilities and capabilities
- facilities and capabilities of other agents (potentially incomplete)

- inter-agent communication facilities
- the current task

Thus the head can be seen as a mediator between the agent's individual functionality and the overall problem solving context. Further, in order for the agent to actively cooperate with other agents, it must have knowledge about the relevant aspects of cooperation such as:

- its own cooperation role(s)
- cooperation roles of agents, with which it is cooperating (this may not be all agents in the c-world)
- globally available cooperation strategies, from which it can choose to implement new cooperation structures
- current cooperation structures it is involved in.

Knowledge of an agent may be determined upon startup of the system (such as a database) or may be acquired dynamically, by transfer between agents in the course of problem solving.

An agent's knowledge can thus be divided into two parts: that which allows it to define its own behavior, at a meta-level, and that used to solve problems or execute tasks [Mah90]. The agent should be able to express meta-knowledge, i.e. a set of knowledge about other knowledge. Meta-knowledge of capabilities, status and behavior of other agents is referred to as *epistemic knowledge*. *Autoepistemic knowledge* pertains to knowledge about an agent's own capabilities. These two types of knowledge make it possible to deal with interaction among agents. In the course of receiving messages, an agent may use its autoepistemic knowledge concerning its capabilities to retrieve information or to accept messages it is interested in, while in sending, its epistemic knowledge allows it to choose the appropriate addressees. Epistemic knowledge can take the form of acquaintances which concern beliefs about interests (goals, plans, etc.) and the capabilities of other agents.

Though application independent cooperation functions are desirable, the requirements will also be driven by the concrete application needs. Thus we will allow for customized tailoring of these features according to the goal and purpose of each individual application. Of further interest is the interface between an agent's head and its body. If the body is an advanced system, the interface will have to be very broad, allowing for the many capabilities of the body. On the other hand, if a body is simple, the interface will be correspondingly less complicated.

In order to communicate with other agents, an agent needs to have access to appropriate telecommunications channels and network information about the

agents, such as their various addresses. The mouth is that portion of the head which fulfills this communication functionality. So the mouth of an agent would basically process a message from the head, find out which address to deliver it to and send the message. It would also receive messages from other agents and send them to the head for further processing. Thus, the mouth represents the interface from the agent to the cooperation world. The mouth also has the responsibility of receiving messages from other agents and passing them on for further processing to the head.

2.2 Knowledge Bases and Inference Engines as Agents

This general agent model enables existing or new knowledge bases to communicate with other agents, such as human users, reasoning engines, or other knowledge bases. This ability is achieved by providing the knowledge base body with a head and a mouth. The head consists of a cooperation knowledge base and other coordination modules, while the mouth serves as the actual communication device. Inference and deduction engines, which usually form the reasoning module in knowledge based expert systems, can be extended similarly. The body functionality of these agents is, of course, different from the knowledge bases. Conventional expert systems consist of a knowledge base and a reasoning engine, either deductive or inference, in one system.

3 Expert System Ad-Hocracies

With cooperating knowledge bases and modularized expert systems, we have the opportunity to configure expert systems for very specific purposes as the need arises and to quickly reconfigure them for the next task. Coordination technology, enabling human task forces and ad-hocracies, thereby breaking up rigid organizational structures and allowing flexible responses to changes in complex environments, can be expanded to the realm of machine agents.

The most basic case is to link a particular knowledge base to a specified reasoning engine. These two agents thereby form a conventional expert system for the time of their cooperation. Important issues in this case are the compatibility of system formalism and the question of initiative for finding a partner. This scenario can be extended to a reasoning engine cooperating with several knowledge bases to solve a certain problem. Questions of compatibility between knowledge base formalisms and aliasing problems arise here. The most sophisticated case would be the cooperation between several reasoning engines, where each engine draws on one or more knowledge bases. Conflict resolution between contradicting findings of the engines are just one of the many issues in this scenario. The fourth

general scenario many reasoning engines cooperating with a single knowledge base, is not too different from the two previous scenarios. The issues concerned with these scenarios will be discussed in the following subsections.

3.1 Matching Reasoning Engine and Knowledge Base

Traditionally, expert systems consist of a reasoning engine and a knowledge base [BBB+83, Nil80]. The reasoning engine is usually tuned to the types of problems the expert system is expected to solve, eg., deduction, induction, means-end analysis, etc. The knowledge base is viewed as a module that carries the static domain expertise. By changing the current knowledge base in an expert system, problems of the same type, eg., fault diagnosis, but in a different domain should be approachable. In a system of cooperating knowledge bases and reasoning engines, where every knowledge base could possibly be connected with any one of the reasoning engines, the conventional expert system paradigm presents the base case and is achieved by connecting a knowledge base and a reasoning engine over a certain period of time.

This most basic case of cooperation between a knowledge base and a reasoning engine can be initiated in many ways. A human user browsing a number of knowledge bases and deciding to use an appropriate knowledge base to work on a certain problem can leave the initiative with the knowledge base. In this case, a reasoning engine needs to be found that is compatible with the knowledge base formalism and that does the type of reasoning the user desires, e.g, medical analysis, mechanical configuration, etc. If, for example, the user wants to diagnose a liver disease and has already decided on a particular knowledge base, containing information on such diseases, an engine must be found that is capable of doing medical diagnosis.

On the other hand, the user could have started from the reasoning engine side, rather than from the knowledge base side. The user has decided on a certain reasoning engine, or class of reasoning engines, by browsing the system or by a focused search By first deciding on the type of reasoning engine desired, the initiative for finding the cooperation partner to form a complete expert system would be shifted to the engine The two cases (knowledge base first, engine first) are symmetric, as far as the communication between the heads of the agents (the knowledge base and the reasoning engine) is concerned. The agent that was initially selected has to find a compatible partner The third case, where the user selects both agents and directs them to cooperate with each other is very similar to the conventional expert system view and does not present many cooperation problems.

To help an agent (knowledge base or reasoning engine) find the partner that is needed to successfully solve the user's problem, knowledge about the agent's own

abilities and needs is used, as well as knowledge about other agents in the world that can possibly cooperate with oneself and the description of the task to be solved. The single most important factor lies in the compatibility of formalisms, though current research is trying to solve the problem of compatibility between different knowledge representation schemas [BKMed]. Histories of previous, successful cooperations and general match-making strategies give the agent looking for a partner a wide variety of cooperation possibilities. If the agent knows about matching partners, requests are send out to these partners. If more than one partner respond, the initiating agent applies conflict resolution strategies to choose the best partner. Factors in such strategies are the degree of match for the problem at hand, the time/load constraints of the partner, the cost of communication, and other less important factors. If neither of the addressed partners replies, the initiating agent puts an advertisement for a job to be done on the net. This gives all agents in the system the chance to participate in the particular task. It also presents a chance for the agents to learn about new agents that joined the system and were not part of their initial, local knowledge about the system. Agents receiving an advertisement decide on the basis of the workload and appropriateness whether to respond or not The initiating agent will apply the same conflict resolution strategies as above in the case of many responses. The worst case consists of no agent responding at all. The human user must then be informed that the problem is currently not solvable. A special case consists of an agent promising to cooperate in the future but being unable to do so immediately because of other duties. In this case the human user can be informed and decide whether to wait or to cancel the query. When two machine agents have agreed to cooperate, they have to define the terms of this cooperation. Time limits, security issues, etc. must be specified. This can take the form of inter-machine negotiation. The result is an apparently conventional, but highly specialized expert system, which will exist only temporarily.

3.2 Beyond Dyadic Cooperation

Although creating highly specialized expert systems on a demand basis is already a large benefit of cooperating knowledge bases, the true power of such systems arises when more than two agents join forces. This is the case when a reasoning engine consults several knowledge bases, rather than merely one, or when several engines work on the same problem These benefits do not come for free though Problems, such as concurrent processes, contradicting derivations, and inconsistent knowledge bases, are just a few of the large set of issues to be regarded Obviously, we can not provide solutions to all these problems, but we can sketch the way such systems could be build and how separate solutions to the above problems could be integrated.

The most pressing problems from the cooperation point of view are those of cooperation strategies. *Cooperation* can be decomposed into a planning component and an execution component. We ascribe the term *coordination* to the joint solving of the problems and planning of action, while joint execution of these plans is labeled *collaboration*. Cooperation strategies can be instantiated idiosyncratically in both realms. In general, we can find two extremes on the spectrum of cooperation strategies: master-slave-cooperation and democratic-participation. A large number of other strategies, such as mutual-observation or consider-leader-proposal-first, lie between these two extremes.

3.2.1 One Reasoning Engine with Several Knowledge Bases

An expert system can boost its power by certain orders of magnitude if the reasoning engine in the system can draw on a larger amount of knowledge. More knowledge means more ways to solve a problem or achieve a goal for the system, or being able to tackle problems that were previously unconquerable. The approaches taken to supply an expert system with a more complete domain model, range all the way to machine learning and expertise transfer. By allowing knowledge bases and reasoning engines to cooperate freely, we open a simple alternative to the above strategies. Instead of linking one reasoning engine with just one knowledge base, many knowledge bases can be made available to the reasoning engine.

The way this cooperation can be brought about is similar to the case where a reasoning engine is actively searching or generally advertising for a knowledge base. When the reasoning engine receives offers from a number of compatible knowledge bases, it can choose among a number of cooperation strategies. Given that the knowledge bases are willing, the reasoning engine can assume a coordinator role and assign passive roles to the knowledge bases, meaning that the knowledge bases will only provide information when queried for by the reasoning engine. A more relaxed scheme allows the knowledge bases to communicate among each other and to detect contradictions before the reasoning engine can reach faulty conclusions.

The advantages for a reasoning engine of having available multiple knowledge bases arises mostly from the fact that knowledge bases can complement each other and that the engine can verify a derivation in several ways. Knowledge that might be missing in one knowledge base can possibly be found in another one. A "knowledge gap" can therefore be bridged by switching to another knowledge base until work with the first one can safely be resumed. This process of employing several knowledge bases to bridge knowledge gaps is not necessarily restricted to two knowledge bases but is easily applied a number of knowledge bases. The second advantage, multiple derivations, also arises from the fact that more knowledge is available to the reasoning engine. A derivation that was justified only

marginally by facts in one knowledge base can gain in plausibility when facts in other knowledge base support the same derivation. This helps in the decision among alternatives and adds to the credibility of the overall system.

Again, these advantages are bought at a cost. While additional knowledge can help to bridge the knowledge gap and lead to additional support for certain derivations, it can also lead to contradictive derivations and aliasing problems. The same propositions might be called by different names in the participating knowledge bases. Thus a fact that can actually help to bridge the knowledge gap, would not be recognized. Cooperation in this case means constructing aliases tables from dictionary knowledge bases. Another class of problems arise, when the same label of a proposition carries different meanings for the participating knowledge bases. The proposition **breaks** could be used transitive and intransitive, as in *John breaks the glass* or as in *the glass breaks*, which have clearly different meanings. This case is harder to resolve than the previous one of merely aliasing via a translation table. In the case of different meanings (proposition overloading), the instances of overloading have first to be recognized and recorded. Then measures can be taken to either rename proposition when used by the reasoning engine, or to decide to stop using one of the knowledge bases involved. Problems can also arise, when different knowledge bases allow the engine to reach contradicting conclusions. Knowledge base "A" might for instance allow the conclusion of "X", while knowledge base "B" supports "not X". Conflict resolution strategies and negotiation policies have to help the reasoning engine to deal with these matters.

3.2.2 Several Engines with Several Knowledge Bases

The most complicated case of cooperating knowledge based systems consists of several reasoning engines cooperating with one or more knowledge bases, forming a distributed hybrid expert system with a large variety of reasoning methods, parallelism, and an extensive domain model. The advantages of such a system are obvious. Alternative solutions to the same problem, shared work load, and highly efficient means end analysis are just a few. These advantages, however, still have a certain price: contradictions, inconsistencies, access rights, and contingencies, to name just a few.

For the moment, we will consider only the coordination problems of such agents. First the question of initiative has to be addressed. As in the cases above, a human user could order one or more agents from the knowledge base and reasoning engine domain to work together and, if necessary, look for partners. This leaves us in the general case with a set of knowledge bases and reasoning engines, which still need to involve other agents to solve the task given by the user. We can view the following coordination process as a set of rounds which never produces a stable solution. First the agents look for the partners needed by themselves. After

the agents have independently found partners, they have to determine the compatibility of all agents involved. This leads to certain partners leaving the group and a tighter set of "partner requirements" being developed. After a number of such rounds, the groups grows more stable and compatible, and certain partners might be able to start solving parts of the task given.

Another possibility is to have a central agent, which serves as a register for available knowledge bases and reasoning engines. An agent, requiring a partner to solve a task issues a request concerning the problem at hand to the register. Based upon its knowledge as to the capabilities of the various other agents in the system, it could match one or more partners to the original requesting agent. An extended role of this register agent would be actual coordination of the agents, specifying who should do which task and recognizing plan overlaps [Stu85].

4 Summary and Future Research

We presented an agent model and a number of cooperation strategies that enable knowledge bases and reasoning engines to form highly specialized expert systems. The benefits of cooperation techniques and technologies from the are of human cooperation are transplanted fruitfully to the area of cooperating machine agents. We are currently implementing this agent model and are starting to formalize cooperation strategies on the basis of coordination science In order that the Knowledge Bases and Reasoning Engines involved in the cooperation process be as heterogeneous as possible, it is desirable to have a formal knowledge representation language (FRL) to be used as a default for communication Each knowledge base/Reasoning Engine would be able to translate the FRL into its own internal representation language. Agents with the same internal representation language could then use that for more direct communication, involving less overhead. Several efforts are being made to establish a FRL [BCM90, MCC90], whether suitable translation engines can be developed remains to be seen

5 Acknowledgements

We would like to thank Hans Haugeneder and the members of the KIK Project for contributing ideas in this paper regarding the agent model. We would also like to thank Bruce Croft, Nehru Bhandaru, and Oddmar Sandvik of the Collaborative Systems Laboratory at the University of Massachusetts for many discussions.

References

[BBB+83] B. G. Buchanan, D. Barstow, R. Bechtal, J. Benett, W. Clancey, C. Kulikowksi, T. Mitchell, and D. A. Waterman. Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors. *Building expert systems*, pages 126 –167. Addison-Wesley, Reading, Massachusetts, 1983.

[BCM90] N. Bhandaru, W. B. Croft, and D. E. Mahling. A Task-Centered Theory of Cooperative Work. Technical Report 90-25, COINS, University of Massachusetts, Amherst MA 01003, March 1990.

[BKMed] Chitta Baral, Sarit Kraus, and Jack Minker. communicating between multiple knowledge-based sytems with different languages. In *Proceedings of CKBS 1990*, to be published.

[Con89] Imagine Consortium. Imagine: Integrated multi-agent interactive environment. Technical report, ESPRIT, 1989.

[Mah90] D. E. Mahling. *A Visual Language for Knowledge Acquisition, Display and Animation by Domain Experts and Novices.* PhD thesis, University of Massachusetts, February 1990.

[MCC90] D.E. Mahling, B.G. Coury, and W.B. Croft. User models in cooperative task-oriented environments. In *Proceedings of the 23rd Hawaii International Conference on Systems Science.* IEEE, 1990.

[Nil80] N. J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann, Los Altos, 1980.

[Stu85] Christopher Stuart. An implementation of a multi-agent plan synchronizer. In *Proceedings of IJCAI 1985*, pages 1031 – 1033, 1985.

[WBH87] J. Whiteside, J. Benett, and K. Holtblatt. Usability engineering· Our experience and evolution. In M. Helander, editor, *Handbook of human-computer interaction.* North-Holland Press, 1987.

Integrating Distributed Expertise

Mark R. Adler
Evangelos Simoudis
Digital Equipment Corporation
290 Donald Lynch Boulevard (DLB5-2/B4)
Marlboro, Massachusetts 01752 USA

Submitted to.
International Working Conference on
Cooperating Knowledge Based Systems
October 3-5, 1990
University of Keele, England

OMNI is a framework which serves to integrate a number of existing, heterogeneous distributed problem-solving components into a system which utilizes the expertise of the component parts to solve a wide range of problems while minimizing the need to alter the individual systems This integration is achieved by creating a "wrapper" around each component system. Thus each component consists of two parts a filter which determines whether or not the component can contribute to solving the current problem, and an engine which solves the problem. Through the use of a distributed blackboard communication model, our framework establishes a single system which integrates the existing "islands" of expertise, and can be easily updated to include more and more specialists as they are independently developed. This paper describes our framework in the context of other work in distributed AI.

1 Introduction

OMNI is a Distributed AI (DAI) framework for integrating existing, heterogeneous, knowledge-based systems that are deployed in a distributed computing environment. OMNI solves diagnostic and advice-giving problems by utilizing the expertise of its component systems.[Bil 90] We integrate otherwise disjoint, problem-solving components without requiring their alteration before being incorporated into the framework. Integration is achieved by associating a decision "agent" with each knowledge-based system (KBS). Each agent determines if the associated KBS component can contribute towards the solution of the current problem.

A "central controller" or distributor serves to coordinate the activity. The distributor utilizes a domain vocabulary of relevant terms to determine which systems may offer solutions. Using a contract-net style communication protocol, the distributor obtains bids from the component systems, and selects appropriate systems to answer the query.

OMNI's heterogeneous component systems may either reside on different nodes of a computer network or on different processors of a multiprocessor computer system. By "heterogeneous" we mean. components that are totally different in terms of domain of expertise, styles of reasoning, implementation language, host computer architecture, and operating system Our initial experiments have dealt with three such component systems· two rule-based systems and one model-based reasoning system. We have used them to diagnose the cause of operating system failures, printer problems, and computer network problems.

2 Example Scenario

Suppose the user of a computer system tries to print the file "baz.ps" on the laser printer called lazer2, and discovers that his output is unintelligible. He wants to find out how to solve his problem. So he types the following question to his computer:

Why do I get unreadable output out of lazer2 when I try to print baz.ps?

Assume that there exist three diagnostic systems. For lack of better name let us call them: Hewey, Dewey, and Louie. The first diagnoses the causes of VMS[1] crashes, the second is an expert system for diagnosing problems with queues, batch jobs, printing, and printers. The third expert system can diagnose network problems. These systems receive the user's question and must determine for themselves whether they can contribute to the solution of the problem.

First, each system, using minimal problem-solving efforts, tries to decide if the problem is within its domain of expertise. During this process Hewey determines that the problem is not due to an operating system crash since it cannot locate a crash dump file (the result of a crash) on the file server. However, the information that has been provided by the user is meaningful

[1] VAX/VMS is a registered trademark of Digital Equipment Corporation.

to Louie since it deals with printers and print jobs Appropriately, Louie informs the user that it will be able to attempt to solve the problem After verifying that there exists a network connection between the printer lazer2 and the file server, Dewey also informs the user that it will be able to attempt solving the problem.

Second, the user evaluates the information provided by Louie and Dewey in the context of what he knows about each of these three systems, and decides to ask Louie to try solving the problem After executing, Louie returns the following text to the user

lazer2 is not a postscript printer. Either select printer PSfoo which is also on your network, or convert your file to another format.

This information satisfies the user's request and they live happily ever after

3 Definitions

Before proceeding, let us define some terms that will be used throughout this paper

- PSM Each of the problem-solving components in OMNI is called a Problem-Solving Module (PSM) A PSM consists of a broker and a specialist.
- Broker The broker is the part of the PSM that determines whether the PSM can contribute towards the solution of the user's current problem. It is responsible for the interface between the PSM and the outside world, and for informing the outside world of the PSM's contribution
- Specialist The knowledge-based system that is integrated with OMNI is called the specialist
- Distributor A separate, central agent that performs evaluation and control functions using a registry of PSMs, a vocabulary of terms, and the data that is generated by the user and PSMs during the problem-solving process

4 The Distributor

In response to a user's query, the distributor issues a request for bids (RFB) to a subset of the PSMs' brokers Each broker determines whether its specialist can contribute to the solution of the user's problem and returns a bid The distributor evaluates the bids and issues an award to the selected PSM

Controlling the operation outlined above involves

- Understanding the user's query and generating an RFB Understanding implies that the user's query will be "seen" in the context of the problem-solving capabilities of each PSM.

- Partitioning the space of PSMs and determining the brokers which will receive the RFB
- Evaluating the received bids, selecting the most appropriate one(s), and generating an award for the PSM which will attempt to solve the problem.
- Determining the next step after a solution is returned from a PSM to the user

Currently we are assuming that the user will be evaluating the quality of the solutions that are returned by the PSMs. This is not an unreasonable assumption since the output of most of the expert systems that exist today in academia and industry is in the form of a textual report. We believe that automatically interpreting and evaluating this output is beyond the scope of this project. However, if the output of the PSMs is provided in a machine understandable form, then in addition to the actions that are listed above we will also need one to evaluate the solutions provided by the PSMs.

The distributor must include the following types of knowledge to accomplish the above actions

- Interpretation knowledge. We are *not* immediately concerned with the issues behind processing the user's query in its natural language form, even though we know that in the long run we will not be able to avoid it. Instead we have focused on the problem of expressing the output of a natural language processor in terms of a vocabulary that is understood by all the brokers which are part of OMNI. Even though we do not currently support direct communication between PSMs we nonetheless allow two or more PSMs to communicate through the distributor in order to achieve cooperation for the solution of a problem We call this common vocabulary the level-0 vocabulary.
- Knowledge about PSMs. This is knowledge about the domains of the PSMs and major tasks within these domains. This knowledge has been organized manually into a task hierarchy The leaf nodes in this hierarchy contain the names of the PSMs that cover and solve a particular task The task hierarchy is connected with the level-0 vocabulary using activation links which allow the distributor to identify the PSMs that will receive an RFB A set of logical operations over the set of activation links is specified to indicate whether each PSM is relevant. A portion of the task hierarchy that is used by the current version of the distributor is shown in Figure 1.
- Evaluation knowledge. In order be able to evaluate the bids that are submitted by the brokers, the distributor has knowledge about the cost of allowing a specialist to try addressing the problem. We measure cost along the following dimensions user interaction requirements, computer resource requirements such as files and databases, runtime requirements, and user's expertise requirements (i.e whether the specialist assumes the user is a system manager, a network communication specialist, or a naive Unix[2] user)

[2]Unix is a registered trademark of A.T.&T.

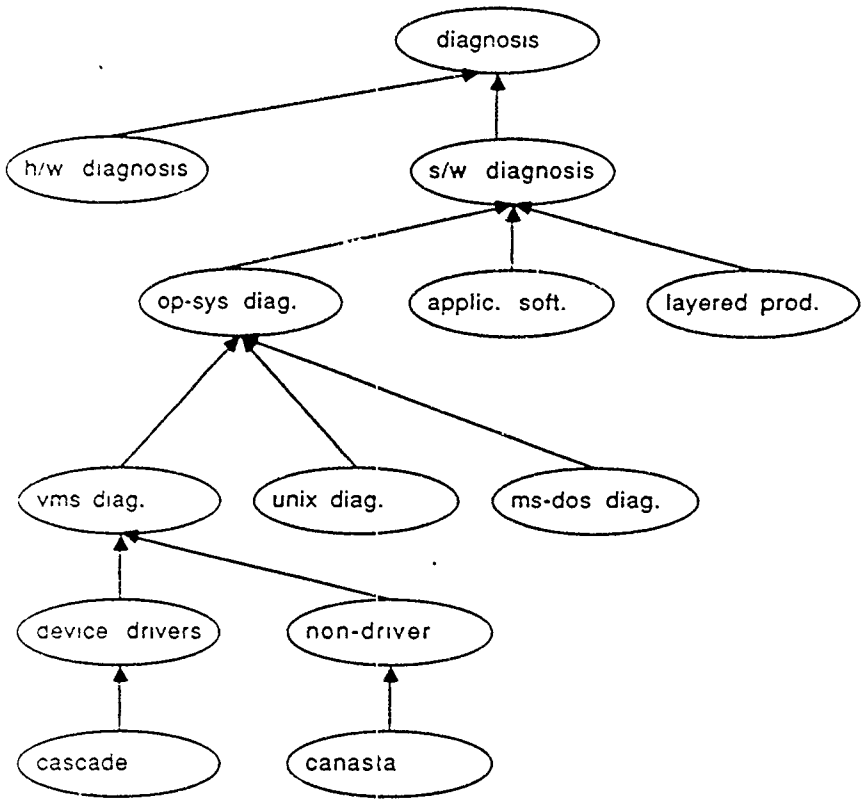


Figure 1: The task hierarchy

- Control knowledge The distributor needs control knowledge to determine how to act when:

  1. A problem is stated by a user
  2. A solution that is provided by a specialist is not satisfactory to the user.
  3. A broker instead of submitting a bid, enhances the description of the problem.

  For the first action, the distributor has to identify the appropriate brokers, and issue an RFB to each. This process is explained in Section 4.2 after we provide more details on the level-0 vocabulary.

  The second action can be addressed in two ways depending on the situation at hand:

  1. If multiple bids have been submitted then the distributor selects one of the remaining ones and issues an award to the appropriate PSM. Alternatively, it can be made by the user.
  2. The distributor requests the user to clarify the query using different language, and repeats its basic control cycle.

  The third action stems from the fact that problem statements are usually ambiguous. The distributor's interpretation knowledge along with its knowledge about PSMs does not allow it to disambiguate the problem. However, a broker, using its own domain-specific knowledge may be able to suggest an interpretation of the problem statement. For example, the problem statement "Is my line too long?" can be used both in the context of text editing and in the context of computer network hardware. As a result, the broker of a PSM for the EMACS text editor may offer the following elaboration of the problem statement: "I believe that you are asking a question about editing a line of text that is longer than your screen can accommodate." On the other hand, the broker of a PSM that can diagnose computer network problems (the Louie of our example above) may offer the following elaboration: "I believe that are asking a question about signal strength in a local area network configuration". Assuming that the user had a way of understanding the submitted interpretations, and that the number of interpretations was manageable[3], then he could decide which was the most appropriate. Alternatively, after reviewing the suggested interpretations the user may decide that none is appropriate, and that the original problem has to be stated differently. In such instances the distributor starts the problem-solving process from the beginning.

[3] this last point requires user modeling because what is considered a manageable number of topics differs among users

4.1 Level-0 vocabulary

It will be very hard to create, and even harder to agree on, a domain-independent, common vocabulary to be used by both the distributor and the PSMs for problem interpretation. Instead, we believe that a more tractable problem is to create a high level, domain-specific, shared vocabulary. For the time being, we have taken an *ad hoc* approach and generated this vocabulary out of concepts that are used in OMNI's initial application domain

The vocabulary consists of terms and relations between them. For example, a level-0 vocabulary for the domain of computer software and hardware diagnosis will include terms such as: operating system, CPU type, disk, application program, etc.

Terms are connected using generic and/or domain-specific relations. We have provided partonomic, categorization, and synonym relations as the generic relations to connect the terms in this vocabulary For example, a *part_of* relation connects the terms "disk drive" and "CPU", an *isa* relation connects the terms "VMS" and "operating system", and a *synonym* relation connects the terms "computer" and "node" Relations can also be domain-specific. For example, we consider the relation *connected_to* to be a primitive relation in the domain of computer software and hardware The relation *allows_access* that is used for connecting the terms "computer account" and "operating system" is an example of a domain-specific relation. Constraints between terms (e.g the VMS operating system requires a VAX CPU) are also expressed as relations

Each term can also be associated with one or more of the leaf nodes that contain the names of PSMs in the task hierarchy This association is established using activation links. Several terms can be linked to a specific leaf node

The level-0 (or base) vocabulary is made available to the broker of each PSM as soon as the PSM registers with the distributor This does not mean that the broker has to utilize this vocabulary in order to respond to an RFB A broker may interpret a user's query using a much simpler scheme such as table lookup Nor does the availability of the level-0 vocabulary imply that all of the domain-specific concepts used by the PSM will already be included. For example, the term "bugcheck", essentially an error code that is issued by the VMS operating system when an abnormal condition occurs, does not belong to the level-0 vocabulary, but it is an essential term for the broker of the PSM that diagnoses VMS crashes. Such a term will need to be local to the broker since it is too specialized for the distributor's needs The existence of terms that are specific to a particular PSM implies that each PSM must have its own vocabulary (let us call it level-1 vocabulary) for interpreting a user's problem Local terms may "shadow" terms in the level-0 vocabulary.

4.2 Query interpretation

As a user's query is being processed, it activates terms in the level-0 vocabulary. Interpreting the query using the vocabulary is performed by trying to establish a path between all the terms that

are activated. A path is established by following the directed arcs that describe the relations between terms. This is performed using a marker-passing algorithm that simultaneously starts from every term activated by the user's query, and follows all the arcs that emanate from those terms

By establishing a path through the appropriate terms, and following the activation links between the terms in the path and the task hierarchy, the distributor determines the PSMs which will receive RFBs. Portions of the level-0 vocabulary and of the task hierarchy are shown in Figure 2.

For example consider the user's query:

"How do I restore the operation of my crashed VMS system?"

The query is parsed, and the nodes labeled "crash" and "VMS" in Figure 2 are activated The marker-passing algorithm starts at these two activated terms, and successfully terminates after the node labeled "error" is reached because there exists a path to this node from both of the starting nodes Therefore, the user's query is successfully interpreted because the initially activated terms have been semantically linked. Next, the distributor identifies the nodes in this path that have activation links to leaf nodes in the task hierarchy It finds that the proper conjunction of links between the terms "VMS" and "crash" and the leaf nodes labeled "cascade" and "canasta" are satisfied The distributor next requests bids from the brokers of both of these PSMs

As was stated above, several of the terms in the level-0 vocabulary may be connected to a particular node in the task hierarchy. For example, the terms "VMS", "crash", and "driver" of the level-0 vocabulary are connected to the leaf node labeled "cascade" in the task hierarchy Now, consider the following user's query:

"Why does the TTDRIVER in my system causes the system to crash?"

As a result of this query, the terms "crash", and "ttdriver" are activated. Four paths are examined by the marker-passing algorithm; three start from the node labeled "ttdriver", and the fourth starts from the node labeled "crash".

The first path goes through the nodes labeled: "ttdriver", "driver", "module", and "operating system". The second path starts at the same node and goes through the nodes labeled: "ttdriver", "VMS", and "operating system", because ttdriver is only part of the VMS operating system The third path goes through the nodes labeled "ttdriver", "VMS", and "error". The fourth and final path goes through the nodes labeled: "crash", "fatal error", and "error".

Since a completed path exists (the third and fourth paths both intersect in the node labeled "error"), the user's query has a consistent interpretation As a result of the established paths, notice that the leaf node labeled "cascade" from the task hierarchy has all three of its activation links satisfied, and the leaf node labeled "canasta", has two satisfied links. The distributor submits an RFB to both PSMs, "cascade" and "canasta".

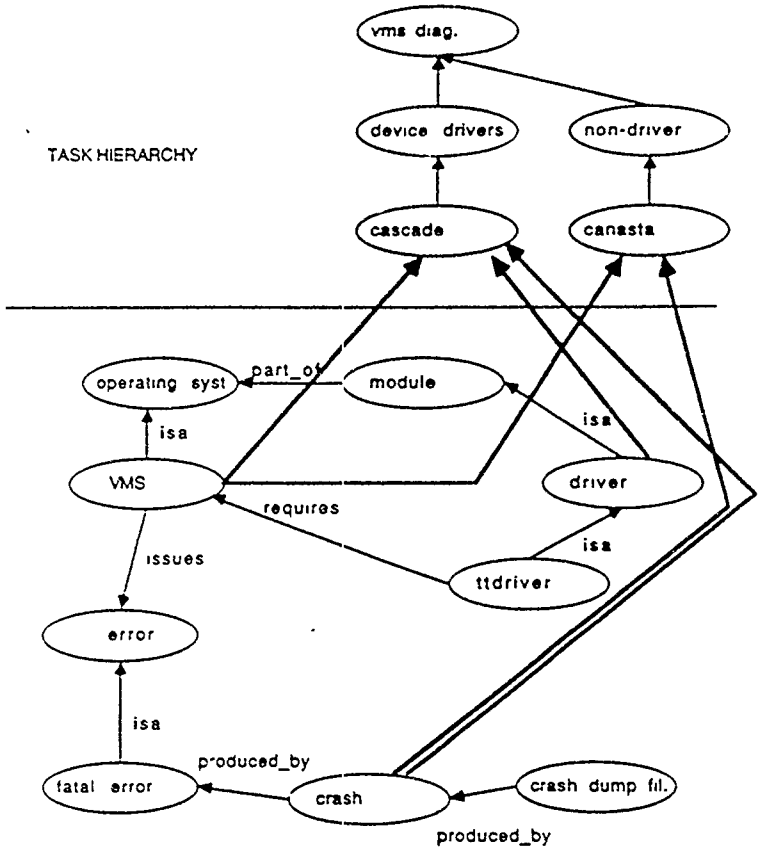


Figure 2 Level-0 vocabulary and task hierarchy

5 PSMs

Our goal was to design an architecture to integrate otherwise disjoint, problem-solving components *without requiring their alteration*. We have achieved this integration by associating a decision broker with each problem-solving component, called the specialist The broker, provides the additional control mechanisms and decision support to allow the specialist to be incorporated into the overall problem-solving mechanism as an independent, modular unit conforming to both the bidding and award cycles We intend to allow any specialist to be incorporated into our framework. We have, therefore, focused on identifying and resolving issues that are associated with the definition and the operation of a broker.

5.1 Brokers

The broker must be able to bid and respond to an award. For bidding, the broker must:

1 translate between the global and the specialist's representation of a problem,

2 interpret the translated representations to determine the specialist's potential contribution to solving the problem,

3. refining the user's query,

4 estimate the cost of invoking the specialist,

5. formulate a bid;

and for responding to an award, the broker must:

1. invoke the specialist, and

2. translate the specialist's output back to the global representation

We have determined that the most important of these actions are: refining the query, and formulating a bid. These are discussed below.

Since user queries are inherently ambiguous, and the distributor has only a high-level perspective of the domain, we require each broker-specialist to interpret the query from its own ego-centric perspective. Thus each PSM may over-zealously interpret the query as one that is covered by its domain of expertise. This is the behavior that we desire. It guarantees that if there is some PSM in our system whose solution domain is relevant to the problem domain, that PSM will respond.

It is expected that the broker's code will be designed to execute within a reasonable amount of time, thus allowing the distributor to receive all the bids that are possible. Furthermore, we assume that the operations performed by each broker require simpler, and therefore less

expensive computations than those of the specialist All brokers execute in parallel while trying to formulate their bids; however, only one specialist is awarded a contract at any one time [4]

Eventually, we would like to provide a "generic" broker that contains the required control mechanisms, but lacks evaluation predicates and a hook into an associated specialist Such an empty shell could minimize the effort required to build a broker Nothing precludes adding such a crude broker as an initial step, and incrementally improving its performance over time

5.2 Interpretation

The broker either uses the same marker-passing algorithm as the distributor on its level-1 vocabulary, or utilizes its own means of interpretation Thus the interpretation of the user's query by the broker may contradict the analysis performed by the distributor The distributor is acting as a manager with only a high-level view of the relevant concepts, the broker has a more detailed understanding of the domain, and may discover that the user's query is not covered by the specialist's knowledge.

The egocentric and overzealous interpretation of each query may result in a flood of irrelevant or unqualified bids to the distributor To some extent the broker may qualify the bid by indicating the confidence in the bid. The distributor may also maintain a history of broker bids and subsequent results to develop its own notion of broker credibility to use in selecting the most appropriate broker-specialist.

5.3 Refining the query

There are two ways in which a user's query may be refined

- The system may engage the user in additional dialogue to encourage a more precise query, or at least one which incorporates terms from the vocabulary so that a consistent interpretation can be established.
- The PSMs can construct individual interpretations of the query, and iteratively refine the query through repeated bidding cycles, until a global interpretation emerges

5.3.1 Additional User Dialogue

There are times when the broker may (perhaps through additional user interaction) refine the query Thus additional parameters must be registered with the distributor This additional information may be used for another RFB broadcast cycle The problem is one of mapping the

[4] This style of control can be changed

broker's local vocabulary to a common central one. Some low-level information may be lost in this transition Enough must remain to relay the domain knowledge.

The original user query may not be precise enough to allow the system to target the appropriate PSM This vagueness may be due to the user's lack of understanding of the domain. By engaging the user in additional dialogue, the original query may be refined and augmented to enable the appropriate PSM to generate the desired solution.

5.3.2 Cascading PSM Interpretation

The OMNI protocol allows each agent that receives an RFB to interpret the user's query from its own point of view, using its own level-1 dictionary. Since terms may be ambiguous, the same query may have multiple interpretations, each consistent within a single broker's interpretation within a level-1 vocabulary. While many PSMs may receive an RFB, only those who can successfully interpret the query and determine that it is within the scope of their specialist's problem solving domain will respond positively.

The fact that PSMs produce an interpretation based on a local vocabulary, and return them to the distributor, enables the involvement of additional PSMs in future bidding cycles Thus a cooperative refinement effort can be achieved if local interpretations activate additional level-0 terms by providing a link between an active and inactive concept that exists in its level-1 vocabulary

This form of cooperation does not require that any PSMs know of the existence of any other, and is analogous to the way that knowledge sources interact in a blackboard environment Each PSM is capable of contributing the missing link to the set of concepts or "interpretation" of the query that will enable the "right" PSM to be invoked.

5.4 Formulating a bid

Assuming that information relevant to the user's query has been transferred, via the RFB to the broker, and that this information has been translated, interpreted and refined, the broker has to estimate the cost of invoking the specialist and determine the contribution that the specialist can make. A broker may request additional information that is either required in order to determine the specialist's contribution, or is a prerequisite of the specialist. The latter type of information contributes to the calculation of the cost of invoking the specialist Two issues important in this stage are (1) collisions: two brokers asking for the same information, and (2) interpretation of existing information: the distributor already has the requested data in a form other than what the broker expects

The broker's bid includes the nature of the contribution that the specialist can provide For example, the bids of the network troubleshooter (Louie in the example of Section 2) may offer to solve the problem by running a network diagnostic, while another PSM may offer to clarify the question by engaging the user in further dialogue.

Finally, an overall description of the computing resources that will be required by the specialist is incorporated into a broker's bid. This information provides a measure of the specialist's efficiency In the event of competing bids from several brokers, this information is used by the distributor to establish a priority ordering. Though the utilization of system resources will vary from one invocation to the next, we have found that statistical measures such as mean CPU time and mean network bandwidth utilization provide an adequate approximation for our purposes An additional parameter that we are considering is the amount of user-interaction required Depending on the type of problem, one may prefer to minimize or maximize the amount of queries to the end-user.

6 Implementation Overview

We have implemented an experimental prototype of OMNI on our local area network using an object-oriented blackboard as the distributor. This system combines PSMs with brokers and specialists written in different computer languages, running on different computer architectures with different operating systems Brokers run concurrently responding to requests for bids. Specialists, however, run serially; only one award is active at a time[5]. The end-user decides if the generated solution is appropriate, or whether another bidding cycle should be initiated.

Even though we have a large repertoire of knowledge-based systems in our corporation we have selected the following four in order to conduct our initial experiments with OMNI:

- a VMS diagnostician
- a network troubleshooter
- a job and print queue expert
- a general problem solver, initially targeted toward Unix neophytes.

6.1 Integration Effort Required

Our goal is to develop a "generic broker" that can be tailored to meet the needs of any specialist To that end, we have implemented brokers for the four systems mentioned above in order to develop experience in OMNI's initial application domain, and to determine the PSM-independent and specific aspects Through this experimentation we have identified that all brokers must provide the following functionality:

- Handle our message-passing protocol· e.g. receiving requests for bids (RFBs) from the distributor, transmitting bids, receiving awards, transmitting final results, etc.

[5] While our protocol supports parallel execution of specialists, we chose to simplify this for now to concentrate on other issues

- Interpreting the user-generated query embedded in the RFB, and determining the appropriate response
- Passing necessary input in the proper form to the specialist, and invoking the specialist
- Obtaining the result of the specialist's actions and reformatting/translating it for transmission to the distributor.

Of these tasks, only the first may be generic enough to duplicate for all brokers. The rest are dependent on the knowledge of the associated specialist and its underlying representation.

The most difficult task for the broker is to determine whether or not the answer to the user's problem or question can be supplied by the associated specialist. To some extent, a generic broker can be provided such that a small number of specific evaluation functions must be added to complete the functionality.

In our initial prototyping efforts, we required about two days to create the first broker, and a diminishing amount for each subsequent one due to code sharing The last effort required only half a day.

6.2 Experimental Results

We have experimented with several different styles of broker-specialist interaction.

- JOQUR - complete integration of broker and specialist code The user's query is modified and refined through an interactive dialogue The system augments the query with additional terms as they are discovered, and recurses for further processing.
- GP - common LISP process, but separate functions to handle the broker functionality and that of the specialist.
- LANALYST - common blackboard inference engine, but separate knowledge-sources for broker and specialist functionality
- CANASTA - complete separation of broker and specialist code The specialist code represents an existing system; the broker system runs as a separate process.

We are trying to refine our notions of what information is required for the broker to make reasonable evaluations of the capabilities of the specialist, and how best to represent that knowledge. In exploring the spectrum of styles, we hope to discover the tradeoffs among the different approaches. In conjunction with this research focus, we are also investigating the more pragmatic problem of constructing an interface between specialist and broker. To date, our effort has been concentrated on developing the broker, and utilizing existing specialists and those currently under development in our group. More work is needed to establish smooth interfaces between broker and specialist, and between OMNI and the user.

7 Relation to Other Work

Most of the research on Distributed AI has focused on the cooperation among a closed set of systems and the communication, planning, and related interpretation problems that must be solved On the surface, the issues addressed by our OMNI framework may seem simpler, since the concerns of inter-agent communication have been replaced by a centralized communication protocol. Our goal to integrate heterogeneous knowledge-based systems for diagnosis and advice-giving problems has led us to focus on the issue of selecting the most appropriate broker-specialist pair from a large collection of such pairs. For this reason, we had to develop a way of characterizing the domain of each specialist, determining the prerequisites for applying the specialist to the problem at hand, and measuring the cost of running the specialist (across several dimensions)

The translation issues have been recognized and addressed in a database framework in [How 89] Our approach differs in that we are not merely interested in sharing data between expert systems but integrating different areas of expertise into a coherent system

The work of Mark on Consul [Mar 80] took a centralized approach to an advice-giving problem and used a KL-ONE knowledge base to map each request to the appropriate tool. Our approach combines centralized control while distributing the eligibility, and problem-solving knowledge The interpretation of the problem description may vary from one broker to the next due to their predilection or bias for viewing problems from their point of view This relates to the work of Star [Sta 88] on boundary objects

The ideas of communication and control are modeled after the work described in [Bon 88] [Yan 88] The notion of brokers bringing their respective specialists to the attention of a control mechanism is related to the contract net model of Smith and Davis [Smi 81] (though our emphasis is on the selection of an appropriate domain specialist to solve the problem, rather than in distributing tasks for parallelism), and the actor formalism of Hewitt [Hew 80]

We have determined that a blackboard style of control and communication among the PSMs is sufficient for our needs Lately, blackboard models have been used more frequently, especially in DAI systems Stemming from the Hearsay-II model [Fen 88], the blackboard model has evolved to support cooperative problem solving including the work on GBB at The University of Massachusetts and BB1 at Stanford [Eng 88] [Hay 86] [Les 83]

8 Conclusions

Existing knowledge-based applications have a number of implementation philosophies, styles of reasoning, and are deployed over a distributed computing environment This style of development and deployment will not only continue, but is expected to accelerate because of the new styles of computing environments that are being used Integration of such heterogeneous components is further complicated by the fact that these components have been designed as

stand-alone systems Through OMNI, we have provided a framework for integrating distributed expertise and making it available to a variety of users

Achieving the desired integration requires the communication between the PSMs and OMNI's distributor We have not been able to propose a principled way of creating a vocabulary that is shared by all PSMs and the distributor and is used for communication purposes We have shown however, how an application-specific vocabulary that is composed of high-level domain concepts (level-0 vocabulary) can be used for

1 the interpretation of the user's problems, and

2 identifying the PSMs that will receive RFBs (through activation links between the vocabulary and the task hierarchy).

We have introduced the broker formalism, established the tasks that the broker must perform, the types of domain knowledge needed for accomplishing them, and the required integration knowledge that allows each broker to interact effectively with OMNI's distributor Of the identified tasks, we have focused on problem refinement, and bid formulation For the former, we have used the level-1 vocabulary which augments the concepts that are present in the level-0 vocabulary, and allows for a more specialized, PSM-specific interpretation of a problem For the latter, we have used a repository of information that is locally maintained by each broker This repository indicates the prerequisites of the specialist, and specifies the resources required to generate a solution

References

[Adl 90] Mark R Adler, Alvah B. Davis, Robert Weihmayer, and Ralph Worrest. "Conflict-Resolution Strategies for Nonhierarchical Distributed Agents" *Distributed Artificial Intelligence, Volume II*, Pitman, 1990

[Bil 90] Meyer Billmers "OMNI- An Heterogeneous Distributed Advice Giver", forthcoming.

[Bon 88] Alan H Bond and Les Gasser. *Readings in Distributed Artificial Intelligence*, Morgan Kaufman, 1988

[Dav 83] R. Davis and R.G. Smith. "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence* 1983, 20., pp. 63-109.

[Eng 88] Robert Engelmore and Tony Morgan. *Blackboard Systems*, Addison-Wesley, 1988.

[Gas 90] Les Gasser and Michael N Huhns. *Distributed Artificial Intelligence, Volume II*, Pitman, 1990

[Fen 88] Richard D Fennell and Victor R. Lesser. "Parallelism in Artificial Intelligence Problem Solving. A Case Study of Hearsay II", *Readings in Distributed Artificial Intelligence*, ed A.H. Bond and L Gasser, Morgan Kaufman, 1988.

[Hay 86] Barbara Hayes-Roth. "A Blackboard Architecture for Control," *Artificial Intelligence*, vol 26, 1986, pp 251-321.

[Hew 80] Carl Hewitt. The Apiary Network Architecture for Knowledgeable Systems In *Proceedings of the 1980 Conference of the AAAI*, pp. 107-117, 1980

[How 89] H C Howard, D R. Rehak. "KADBASE: Interfacing Expert Systems with Databases", *IEEE Expert*, vol. 4, no. 3, Fall 1989, pp 65-76.

[Les 83] V.R. Lesser and D.D Corkill. "The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks," *AI Magazine*, vol. 4, no. 3, Fall 1983, pp. 15-33.

[Mar 80] William Mark "Rule-Based Inference in Large Knowledge Bases," *Proceedings of the 1980 Conference of the AAAI*, pp.190-194, 1980

[Smi 81] R.G. Smith and R. Davis. "Frameworks for Cooperation in Distributed Problem Solving," *IEEE Trans. on Systems, Man, and Cybernetics*,1(Jan. 1981), pp. 61-70.

[Sta 88] Susan L. Star. "The Structure of Ill-Structured Solutions. Boundary Objects and Heterogeneous Distributed Problem Solving," *Eighth AAAI Conference on Distributed AI*, 1988.

[Yan 88] Ju-Yuan David Yang, Michael N. Huhns, and Larry M Stephens, "An Architecture for Control and Communications in Distributed Artificial Intelligence Systems" *Readings in Distributed Artificial Intelligence*, Morgan Kaufman, 1988.

CONCURRENT INTERROGATION OF DISPARATE KNOWLEDGE-BASES

Patrick O. Bobbie

Division of Computer Science
University of West Florida
Pensacola, FL 32514-5750, USA

Abstract Knowledge-Based (KB) systems, typically, are developed using some expert system shell or conventional development languages (where the underlying deductive and inference or computational model is akin to formalisms of logic). Thus, different KB systems may be implemented using different knowledge representation schemes. KB systems are usually tailored towards specific applications. Rarely does a single KB system support the diverse corporate applications. However, the need and advantages of integrating the diverse corporate information sources is widely acknowledged. The interoperability of different information sources is thus quintessential to the future growth and flexibility of corporate operations. In this paper, techniques for accessing distributed (or locally separate), heterogeneous knowledge-based (HKB) systems, without hindering the simultaneous access to local (native) representations, are addressed. The techniques are based on the concurrent interrogation of disparate knowledge bases using a generic object-based representation as a unifying front-end for cooperation.

1. Introduction

1.1 The Problem

Traditionally, the coexistence of conventional databases with different or similar underlying data models and languages has been (theoretically, in part) made possible via structural, operational, constraint, and language mapping schemes [11]. Some of the mapping schemes, e.g., structural mapping, are facilitated by the similarity of the data structures used for representing and implementing the databases. For example, transforming the structure of the network model to an equivalent structure of a hierarchical model capitalizes on the similarity of the underlying tree-oriented data structures - nodes and arcs are used to represent records and relationships, respectively.

A significant amount of research has also been focused on the development of data language translators for making conventional databases interoperable. In general, however, if there are x different data models or representation schemes, it would require x^2 translators to make the x systems completely interoperable. A commonly employed approach to the multi-model transformation problem is the use of a global database schema (independent of the native data languages) onto which different schemas are mapped [11]. The solution has been extended variously using an unifying object-based *concept schemas* for mapping heterogeneous databases [6], [9], [10]. Also, using object-based schemas requires the mapping of local languages into a common global data language, e.g., an object-oriented language paradigm. Thus, even under the concept schemas technique, each data model requires four translators: two for mapping between local and global data languages and two for mapping between local and the object-based schemas.

1.2 The Methods

By providing a generic language as a common framework for expressing user queries, local data languages would have to be mapped into the generic scheme and vice versa. Thus, the number of translators required is twice the number of different data models, which is considerably less than the x^2 or four times as many; as discussed in the foregoing. The generic language approach alleviates the problem of transformations of local schemas. In other words, there is no concept of global schema or direct exchanges of objects between various local schemas.

Having different data models with a common data language, yet, presents a problem if data sharing is a requisite of the data processing applications. Thus, the inter-operation between disparate databases would require each local inquirer to learn and understand other schema definitions in order to access remote data. However, users are more productive, competitive, and feel encouraged to use different systems when the underlying tool disparities, or idiosyncrasies, are hidden. Hence, the database management system must support transparent data sharing. The methods discussed in the paper illustrate the transparency of data sharing among disparate data sources.

1.3 Summary

The introductory section is concluded with an outline of the focus and organization of the paper. In the foregoing, the problems of integrating or making disparate conventional databases interoperable have been presented. In rest of the paper, the focus is on different knowledge-based systems rather than conventional database systems. Generically, however, the methods presented are also applicable to the integration or inter-operation of conventional databases. The general approach entails the cooperation between different KB systems using a generic object-based language as an unifying front-end. In addition, a technique has been developed to reorganize various knowledge data into related, distributable clusters to facilitate concurrent query execution across the HKB systems. The knowledge data clustering technique compensates for the query translation and transparency overheads. The rest of the paper is organized as follows.

Section 2 focuses on the clustering techniques which is fundamental to the concurrent access of the disparate knowledge data. First, an architectural description of a family of HKB systems is presented. Next, a scenario-styled example is given to explicate the HKB software architecture, followed by techniques for reorganizing the, e.g. rule-based, knowledge data. Finally, examples of knowledge-base clustering are presented.

Section 3 is on the abstract-level specification of the syntax and semantics of the generic

object-based language. An example query is given to motivate the ensuing discussions and to convey the expressiveness of the language. Methods for isolating parts of user queries into distributable subqueries to facilitate concurrent executions, are also discussed. In the last section, a summary of the methods discussed in the paper and the current status of the research, is presented.

2 Disparate Knowledge-Base Cooperation

In the following, the HKB architecture which depicts the components of the system for constructing and processing user interrogations against the disparate, but cooperative knowledge-based systems is discussed. A scenario is presented to illustrate the steps that are carried out to execute parallel (sub)queries across the knowledge-based systems. The methods for organizing and clustering knowledge data to support parallel execution of subqueries, are also discussed.

2.1 The HKB Architecture

Figure 1 depicts two interconnected knowledge-based systems having two different data representation schemes. In Figure 1, suppose User1 and User2 issue local queries expressed in the common, generic object-based format. (See Subsection 3.2) The local decomposition and translation tools transform the queries into locally and remotely satisfiable subqueries for concurrent and effective execution. Methods for clustering knowledge data have been developed such that decomposed queries are efficiently mapped onto clusters of related local knowledge data to achieve improved response time.

Queries are decomposed into four subqueries based on the structural components of the query objects. To reinforce the methods discussed in the rest of the paper, let the data associated with System1 in Figure 1 be a rule-based representation. Similarly, let the KB representation scheme associated with System2 (which is architecturally similar to System1 in Figure 1) be a frame-based or another. In the following, a scenario is presented to illustrate the sequence of query transformation and execution steps that occur when User1 and User2 at System1 and System2 issue queries, respectively.

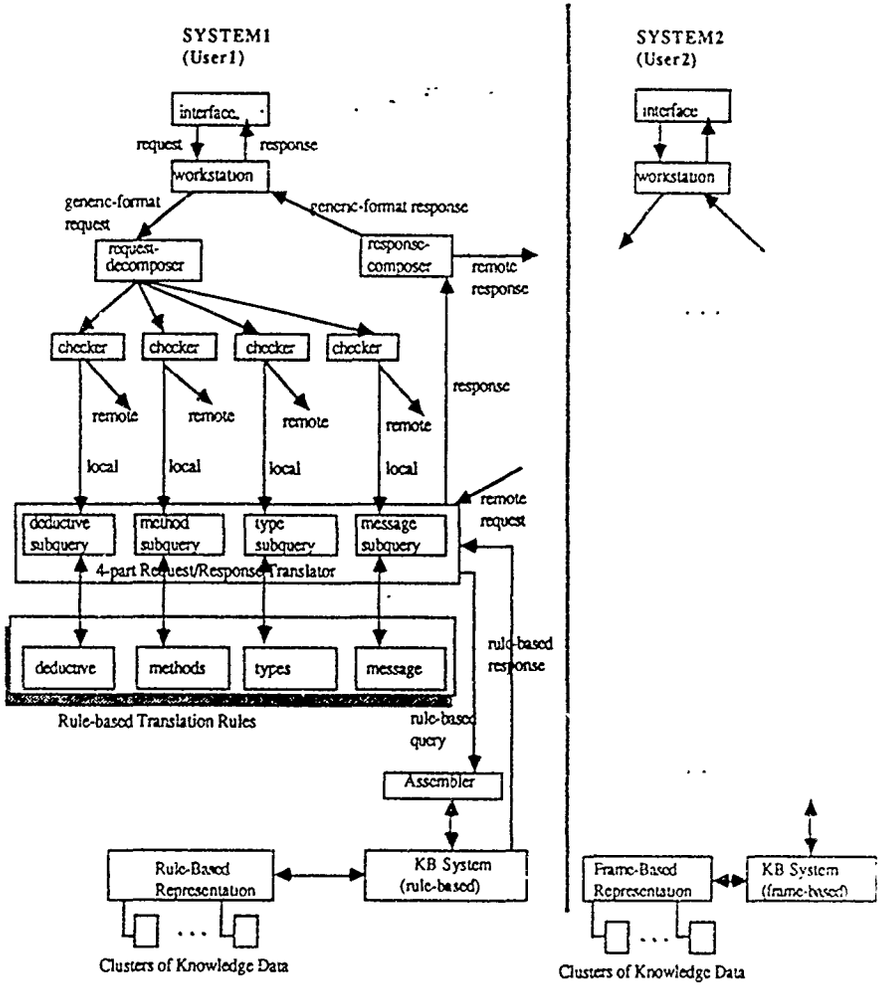


Figure 1. Mapping Generic Queries Between Rule- and Frame-Based KB Systems

Step1: A request is formulated by User1 (at System1).

Step2: The request is decomposed into four parts:

a) the *deductive-constraint* subquery

b) the *method* subquery

c) the *type* subquery

d) the *message* subquery

(Note: the declaration and description information in the query is used to reconcile the differences in the parameter and procedure types and decide on local versus remote executions. See Section 3.2.)

Step 3: The *checkers* determine if the subquery contains information (pertaining to the *method*, *type*, *message*, and logical *deductive-constraint*) which indicates whether the execution is local, remote, or both. (Note that subqueries may be locally or remotely satisfiable.)

Step 4: If a subquery is remotely satisfiable, the subquery is forwarded, e.g., to System2.

Step 5: If a subquery is locally satisfiable, the request/response translator of System1 translates the subquery(ies) using the translation rules: generic --> rule-based *type*, *method*, *deductive-constraint*, and *message*.

Step 6: The translated versions of the subquery(ies) (some of which may arrive as responses from remote-sites, e.g., System2) are assembled by the *Assembler* and passed on to the KB system (rule-based) of System1.

Step 7: The KB system of System1 executes the translated subqueries against the rule-based representation.

Step 8: Any responses are passed on to the *Response-Composer* which employs the 4-part translator (or parts thereof) to translate and forward the response to User1 in the generic object-based format. Remote responses are forwarded accordingly, e.g., to System2.

Step 9: If User2 (of System2) issues a generic object-based request which is locally satisfiable, the request would be executed locally. However, if the request is remotely satisfiable (e.g., satisfiable at System1), the 4-part translator of System1 will receive the subquery(ies) from System2, translate, and execute against the rule-based system. Any responses would be sent to the *Response-Composer* of System1 and remotely dispatched to System2 to continue the local execution.

Step 10: When done, the *Response-Composer* of System2 will forward the generic format to User2. Remote responses are forwarded, e.g., to System1.

2.2 Data reorganization and clustering

2.2.1 The advantages of data clustering

It is widely acknowledged that knowledge-base searching is a time-consuming process which presents a bottleneck in the general evaluation or performance of expert systems. So far, very few methods have been devised to ameliorate the bottleneck. Some of the methods involve parallel evaluation techniques that take advantage of the

* parallelism in the representation language,
* availability of parallel processors or multi-processors,
* [illegible] of multi[illegible]g or s[illegible] of o[illegible]g syst[illegible]ocesse
* some combination of the above.

On the other hand, our findings indicate that research effort in the reorganization and clustering of knowledge-base data objects has been limited. An extensive research into methods, models, and algorithms for analyzing and clustering knowledge-base objects using attributive and behavioral information has hitherto been reported [3], [4]. The clustering methods in [3] and [4] produce both dependent and independent clusters of data objects. Reorganizing and clustering knowledge data has several advantages, among them are:

* The data objects in each cluster are closely related because they may have common functions, behavior, or attributes (example criteria for organizing the objects).
* Searches and accesses of objects within clusters are localized due to the reorganization criteria and less time-consuming (since each cluster represents a fraction of an entire knowledge-base).
* Independent clusters of knowledge data can be processed on separate processors, if available, to improve the overall performance of the knowledge-base management system.
* Independent subqueries can be mapped into relevant clusters and/or evaluated by separate processors or processes.

2.2.2 Knowledge data clustering

In the sequel, two simple examples of knowledge data and representations are presented to demonstrate the cooperative, concurrent execution of subqueries. Appendices A and B are rule-based and frame-based knowledge data representations, respectively. The following assumptions are made about the examples. Let the data in Appendices A and B be associated with System1 and System2 (see Figure 1), respectively. The corporate information represented by both System1 and System2 pertain to employees' family data, manufacturing procedures, safety requirements, products, and locations. The basic difference in both systems is that the knowledge data in Appendix A deals with "agriculture-based" products, while the data in Appendix B is related to "skiing" products. The rule *plant*, which is boxed out in Appendix A, is assumed to have been represented as a frame object and, thus, located at System2.

Figure 2 shows a bipartite graph model of the rules in Appendix A. A bipartite graph is a useful tool for *matching* and reorganizing sets of objects into subsets in which the objects exhibit pair-wise relations. In this example, the knowledge-base rules are viewed as the objects under discourse. Thus, using the bipartite graph representation, the rules are described by the relation, *calls-the-rule* (a dependency relation). The arrows in Figure 2 indicate the pairs of rules that are bound by this relation and the direction of dependency is so indicated by the arrowheads.

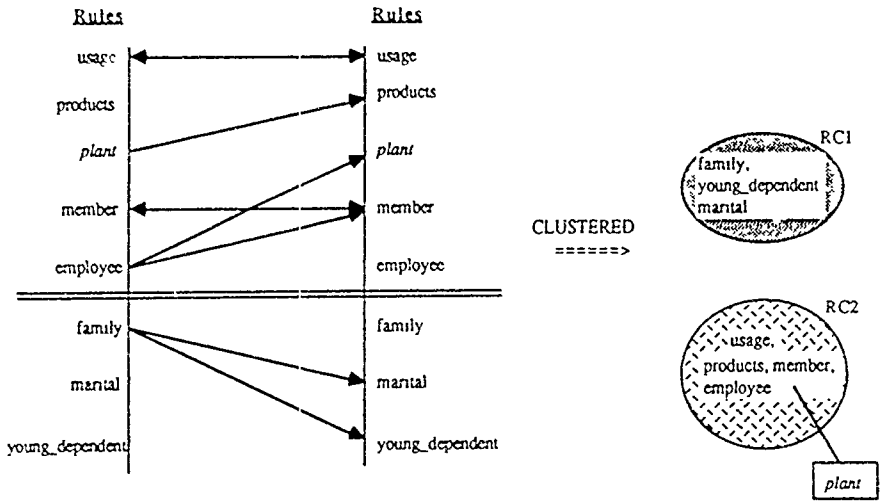


Figure 2. A Bipartite Graph Model of the Rules in Appendix A and the Corresponding Clusters

Legend. ——→ *Calls-the-Rule* Relation

Clearly, the bipartite graph can be split into two independent subgraphs without "cutting" through any arrows, resulting in two non-overlapping clusters of *rules*. The associated *facts* (not defined in Appendix A) are stored along with the corresponding rules in each cluster. For very large rule-based knowledge-base, the rules are modeled as a binary matrix - isomorphic to bipartite graph, and partitioned according to a partitioning algorithm developed in an earlier research [2], [4]. The resultant clusters facilitate parallel evaluation of rule-based subqueries by directing the subqueries to different processors (assuming the data in each cluster is stored on a different processor). Figure 2 also shows the two clusters, namely RC1 and RC2. Since the rule *plant* is assumed to have been represented as a frame object, it is shown in the graph for completeness. The corresponding cluster, RC2, also shows the rule *plant* as an external object. Again, it is assumed that the two clusters are distributed over two processors, available at System1, to facilitate concurrent interrogation/execution of subqueries.

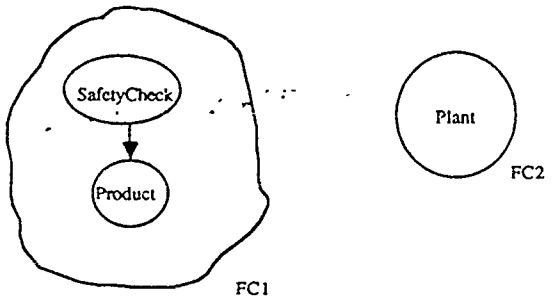


Figure 3. Frame Clusters, FC1 and FC2, of Appendix B

Figure 3 is a collection of clusters, where each cluster encapsulates a frame object definition. The reorganization of frame objects is greatly simplified because a *frame* is a self-contained abstract data structure [6]. Appendix B is a partial listing (three frame objects) of the complete skiing-based knowledge-base data of System2. The frame object, *SelfCheck*, references the *slot*, "ski-size," defined in the frame object, *Product* (local to System2). Thus, the slot "ski-size" creates an object dependency constraint, making it necessary to link and reorganize the two frame objects into a single cluster, FC1. On the other hand, the frame object, *Plant*, does not have a slot, constraint, or action dependency and therefore remains as a single cluster, FC2. Clustering frame objects facilitates future updates of slot-values, action (re)definitions, and constraint evaluations by localizing these operations in a few clusters.

3 An Object-Based Manipulation Language

We begin this section by placing knowledge data representations and manipulation language(s) for accessing data in the following contexts.

* Each KB system is locally independent with respect to representation language, format, structure, and constraints, e.g., rule-based, frame-based, or object-based
* Each KB system has a front-end query construction interface, or subsystem, for developing queries in the generic object-based language.

Queries are issued against both local and remote knowledge data in a transparent manner. Thus, the required cooperation and communication between the different sites is accomplished by the HKB management system (a collection of software modules - see Figure 1).

3.1 Object-based Query Representation

The generic object-based paradigm follows the conventional object-oriented programming syntax

and semantics. Thus, in addition to constructors provided for object data declaration, method definition, and message-mechanisms; mechanisms for object description, deductive-constraints, and type-specific information (of the data objects) are introduced. The syntax of the generic query expressed in an extended object-based format is shown in Figure 4a.

OBJECT *query*	
Description:	target representations
Message:	(sender-message-action) - protocol
Declare:	local data and procedures
Type:	corresponding type and format specifications
Deductive-Constraint:	definition of rules governing method export and data integrity
Method:	specification of exportable and non-exportable action(s)
END *query*	

Figure 4a. The structure of an object-based generic query

Typically, a user query is viewed as an *active* object which has its own attributes. Data objects, transferred between local knowledge-bases, are also viewed as inactive or *passive* queries. The activeness or passiveness of query objects is indicated by the kind of method defined in the query object. This approach for representing both actual queries and data objects uniformizes the manner of knowledge-base interrogation. Thus, the use of query objects as a common mechanism for interrogation, communication, and general data operations parallels the semantics of actor-based systems [1], [7], [8]. In addition, a single mechanism for data accesses simplifies the design of the communication and translation modules of the HKB management system. To understand the semantics of the generic query expression in Figure 4a, an example of how queries are decomposed and executed is presented next.

3.2 Query Formulation

Consider the following query (expressed in a conventional data language format):

```
Q1: if (RETRIEVE emp.name, company.loc WHERE
       emp.status = "single" & company.product = "fertilizer")
    then
       RELOCATE emp.name WHERE company.product = "skiing-apparatus"
```

The intuition behind this query is to determine if there are unmarried men and women employed in the branch of a company where fertilizers are produced (in the farming or agriculture-based region). The expressive power of the query is inherent in the "side-effectiveness" of the antecedent. The additional retrieval knowledge, if a

truth-value of the antecedent also enacts the implementation of the consequence, thus, relocating all unmarried employees in fertilizer-producing region to regions where skiing-gear are manufactured. Typically, it can be assumed that the knowledge-base information for executing the antecedent might be differently represented (e.g., as a rule-based system) and separately located from the data (e.g., a frame-based system) pertaining to the consequence due to the functional or geographical distribution of the corporate data. In such situations, cooperation between the two knowledge-based systems is the norm rather than the exception in meeting the company's data processing needs.

Suppose the above query is posed by a user, the query is first expressed in the generic object-based format using a query constructor in the user's workstation. (See Figure 1.) Parts of the query which are locally satisfiable (depending on the object-operation content) are partially executed against the local knowledge-based system. For the parts which are remotely satisfiable (determined by checking the query *description* representation), **message-stub** subqueries are automatically generated by a local request-decomposer and sent to remote sites for non-local (sub)query satisfiability. Figures 4b and 4c are illustrations of constructed and message-stub queries, respectively. In the sequel, the components of the queries in Figures 4b and 4c are briefly discussed.

OBJECT *query*	*/* query object */*
Description	local:RETRIEVE, remote:serverportid.RELOCATE;
Message	REQUEST(emp.name, company.location) -> RETRIEVE;
	REQUEST(emp.name) -> RETRIEVE + RELOCATE;
	SEND_REQ(RELOCATE(remote-constraint)) -> client;
Declare	emp.name, company.location : ***string***;
	product.code, emp.status : ***string***;
Type	RETRIEVE(*) <-> RELOCATE(*);
Deductive-Constraint	local-constraint => ((emp.status = "single") &
	(product.code = "fertilizer"));
	remote-constraint => ((product.code = "skiing-apparatus"));
Method	RETRIEVE = { (local-constraint = TRUE):
	retrieve emp.name, company.location }
	GET_REP(RELOCATE) = {SEND_REQ(RELOCATE,
	serverportid, messgbuff, messgbufflen, type,
	remote-constraint, repbuff, repbufflen) }
END *query*	

Figure 4b. The structure of an example object-based query

In Figure 4b, the query description entry is an expansion of the original query (as posed by the user). Two factors are employed in determining the location where a (sub)query has to be satisfied, namely the location of the object(s) and the operation(s) as expressed in the query. The selection of *local* versus *remote* execution or migration of objects and operations is performed by the query decomposer. The descriptions in Figures 4b and 4c indicate selections based on operations. The message entry indicates the protocols or the specifications of both local and remote operations, similar to the principles of abstract data type specifications in object-based programming languages. The declare entry defines the data typing or attributes of the objects' parameters. The type entry is fundamental to HKB cooperation, in that the entry specifies, in an abstract manner, the required conversions of data types associated with the operations on the objects between the disparate knowledge-bases. The deductive-constraint entry introduces the concept of embedding rule-action mechanisms in the object-based query to handle conventional query-constraint and integrity problems. Two types of deductive-constraint subentries are provided: constraints associated with both local and remote (sub)query satisfiability. Lastly, the method entry specifies both the local operations and inter-HKB systems communication protocols, as well as the associated constraints.

OBJECT ***query*** ***:message-stub*** , ***or :data object***	
Description	local:RELOCATE;
Message	REQUEST(emp.name) -> GET_REQ; SEND_REP(confirm) -> client;
Declare	emp.name: ***string***; confirm : ***boolean***;
Type	RELOCATE(*) -> type;
Deductive-Constraint	local-constraint => (remote-constraint);
Method	GET_REQ = { (local-constraint = TRUE): relocate emp.name, company.location} SEND_REP(RELOCATE) = {SEND_REQ(RELOCATE, clientportid, messgbuff, messgbufflen) }
END ***query***	

Figure 4c. The structure of a message-stub for a remote query

Figure 4c is a message-stub sent to the remote server for execution of the subquery RELOCATE. The RELOCATE operation is thus considered locally satisfiable after the message-stub is received at the remote site. A remote data management system receives REQUESTs in an asynchronous (polling) mode. When a message arrives, the request is mapped onto a GET_REQ protocol. By default, a SEND_REP protocol is constructed to return

confirmatory messages, or results, to the client. The SEND_REP protocol is, in turn, mapped onto a SEND_REQ procedure to which a counterpart GET_REP is enacted in the client's code. The *Type* entry in Figure 4c enacts a type-conversion of the parameters passed via the RELOCATE request. The *Deductive-Constraint* entry causes the original remote-constraint parameter to be used as a local-constraint at the remote site.

3.3 Concurrent Query Execution

Figures 4b and 4c are the object-based, formulated subqueries to be executed at System1 and System2 respectively. In order to execute the subquery in Figure 4b (local to System1), the subquery is translated into the following rule.

local RETRIEVE ::

retrieve(Plant,X) :-
employee(X),
not(**family**(X)),
products(fertilizer,Plant),
retract(*name*(X)), **retract**(*address*(X)),
system(message-stub(X), serverportid,ReturnCode).

The rules or primitives (in boldface) are defined in System1. The facts (in italics) are also defined in System1, assumed, and clustered along with the corresponding rules in System1. The system-call, system, is a communication primitive for passing the message-stub (as defined in Figure 4c), where the variable *emp.name* is assigned the value of X.

The local evaluation of the subgoal, *employee*, at System1 would require the cooperative and concurrent execution of the rule, *plant* (see Appendix A), which is defined in the knowedge-base of System2 as a frame object. A message-stub (not shown in the paper to avoid cluttering) for the query object, *plant*, is formulated and sent to System2 to be executed against the data in cluster, FC2 (see Figure 3). The results of interest, which are the names of the manufacturing plants in the company, are returned to System1 to complete the local query execution.

From the previous discussions, the knowledge-base at System1 is organized into two distinct clusters as shown in Figure 2. Therefore, the subgoals, *employee*(X) and *family*(X) can also be concurrently satisfied (using two processors or processes) at System1 since there is no *calls-the-rule* relation between both clusters. However, the commonality of the argument, X, would require the application of a set-intersection operation to be performed on the sets of X-values returned by both subgoals. This kind of parallelism in subgoal evaluations is referred to as AND-parallelism and the issues appertaining to such logic-based inferences are well discussed [5].

The subgoal *products* is locally satisfiable at both System1 and System2, hence, only query data objects, with proper type-conversions, would have to be transmitted. Finally, at System1, a system-call is issued to System2 to effect the relocation of unmarried employees to skiing-gear producing plants of the company. At System2, the relocation message becomes a local query.

Finally, at System2, upon updating the knowledge-base, the KB system returns a completion-code to System1 to complete the overall user interrogation.

4 Summary and Conclusion

The foregoing scenario demonstrates the feasibility of making HKB systems in different representations interoperable. The methods for knowledge-data clustering are completed and reported in the literature. Particularly, the approach employed in first reorganizing different knowledge data objects into clusters to facilitate the efficient execution of subqueries is operational and tested on T800 Transputer systems. Currently, the object-based data language and the associated translators and the user interface are being fully developed into a prototype system. In sum, the clustering scheme is a novel parallel query execution scheme which compensates for the inter-HKB systems communication required for remote subquery executions.

References

[1] Agha, G., ACTORS: A Model of Concurrent Computation in Distributed Systems, The MIT Press, Cambridge, MA, 1986, 144 pp.

[2] Bobbie, P. O. and J. E. Urban, "A Model for Understanding the Complexities of Developing Large Scale Software," *1st International Symposium on Tools for AI*, Herndon, VA, October 23-25, pp. 16-23, 1989, also to appear in *Journal of Data and Knowledge Engineering*.

[3] Bobbie, P. O., "Grouping knowledge-base data into distributable clusters," *Knowledge-Based Systems*, Butterworth Scientific Pub., Surrey, England (to appear).

[4] Bobbie, P. O. and M. P. Papazoglou, "Clustering PROLOG Programs for Distributed Computations," *Journal of Systems and Software*, Amsterdam, North-Holland (to appear).

[5] DeGroot, D., "Restricted AND-Parallelism and Side Effects in Logic Programming," *Parallel Processing for Supercomputers and Artificial Intelligence*, K. Hwang and D. DeGroot (eds.), McGraw-Hill, New York, pp. 487-522, 1989.

[6] Feldman, P. and D. Miller, "Entity Model Clustering: Structuring a Data Model by Abstraction," *Computer Journal*, vol. 29, no. 4, pp. 348-360, 1986.

[7] Hewitt, C. and P. de Jong, "Analyzing the roles of descriptions and actions in open systems," *Proceedings of the National Conference on AI*, AAAI, August, 1983.

[8] Hewitt, C. E., "Viewing control structures as patterns of passing messages," *Journal of AI*, vol. 8, no. 3, pp. 323-368, June 1977.

[9] Papazoglou, "Unraveling the Semantics of Conceptual Schemas," Private Correspondence.

[10] Stepp, R. and R. Michalski, "Conceptual Clustering of Structured Objects: A Goal-Oriented Approach," *Artificial Intelligence*, vol. 28, pp. 43-69, 1986.

[11] D. C. Tsichritzis and F. H. Lockovsky, Data Models, Prentice-Hall, New Jersey, 1982.

APPENDIX A

Agriculture-based knowledge data

```
member(_, []).
member(X,[HIT]) :- member(X,B).

family(Z) :- name(Z), marital(Z,Numdep),
             young_dependents(Z).

marital(F,T) :-  if spouse(F,Y)
                  write(Y), nodependents(F,T).
                 ;  /* else */
                 fail.

usage(_, [], _).
usage([], [], _).
usage([A|T1], [B|T2], D) :- kind_of_ingredient(A,K),
                 if not(process_step(A,B))
                  usage(A,T2, D),
                 ; /* else */
                 department(A, C), usage(T1, D, D).

plant(Name, S, Z, D) :- products(S, Name),
        rainfall(R, Name), snowfall(F, Name),
         locations(Name, Z), divisions(Name,D).

employee(X) :- name(X), address(X),
               plant(A,B,C,D),
               department(X,Y), member(Y,D).

young_dependents(D) :- child(K,D), age(K,A),
               not(greater(K,14)).

products([X|Y], F) :- type(X), ingredients(X, [C]),
               stages(X, [Z]), usage([C], Z, Z),
               products(Y, F).
products([], _).
```

APPENDIX B

Ski-based knowledge data

```
FRAME SafetyCheck
slot1:       ski-type
slot2:       ski-width
constraint:  ski-width > 6 inches
action:      defective
constraint:  ski-size < 8
action:      reject
END

FRAME Product
slot1:       ski-type
slot2:       sticks-type
slot3:       ski-size
slot4:       sticks-length
slot5:       department
END

FRAME Plant
slot1:       plant-name
slot2:       {products}
slot3:       {locations}
slot4:       average-snowfall
constraint:  average-snowfall < 10
action:      reduce production by 20 percent
slot5:       average-rainfall
constraint:  average-rainfall < 10
action:      increase production by 20 percent
             END
```

Multi-Agent Expert Systems

Naser S. Barghouti* and Gail E. Kaiser†

Department of Computer Science, Columbia University
New York, NY 10027
naser@cs.columbia.edu,(212) 854-8182
kaiser@cs.columbia.edu,(212) 854-3856

September 3, 1990

Abstract

We investigate the scaling up of a class of expert systems called rule-based development environments, which support software developers in carrying out the software development process, from assisting only a single developer to assisting teams of developers cooperating on a large project. We explore the problem of synchronizing the rule chains fired by the expert system on behalf of these developers while still allowing cooperation among them. In this paper, we outline the three components of our proposed solution. An expanded version has been submitted to *IEEE Expert*.

1 Introduction

Rule-based development environments (RBDEs) model the process of developing a software project in terms of expert system-like rules. RBDEs assist in the development process of a particular project by applying forward and backward chaining on the rules in order to automatically perform some of the chores that developers would have otherwise done manually. The RBDE model has become popular [Per89], and a number of environments based on this paradigm have been proposed and constructed (e.g., [CLF88, MR88, HL88]).

However, existing RBDEs do not scale up to real software development projects involving teams of cooperating developers who share expertise, knowledge and data to perform development tasks that require their concerted efforts. *Cooperation* among these developers requires that they be allowed to concurrently access the project database, in which the shared data and knowledge reside. In order to effectively assist in the development of real projects, RBDEs, which treat the common project database as the working memory of the rule system, must support cooperation among the developers while maintaining the consistency of the project's data. Unfortunately, the current expert system paradigm, on which RBDEs are based, lacks appropriate synchronization mechanisms through which multiple *agents* (either human users or rules) can concurrently access the working memory of an expert system without rendering the data in it inconsistent.

There has previously been some successful research on cooperation among multiple expert systems, each with its own rules and separate working memory — typically with communication through a blackboard architecture [Nii86]. There has also been interest in parallelizing expert systems to improve performance. But there has been little consideration of the problem of *synchronizing* the simultaneous access of multiple rule chains to the contents of a shared working memory while multiple human users are also updating the working memory. Sharing of the working memory, which represents the project database, is necessary for cooperation among the members of a large software project.

The problem of concurrent rule executions has been addressed previously in the context of speeding up the execution of production systems through the use of parallelism. Researchers have concluded that the speedup from parallelism is quite limited in the context of OPS5 production systems that are implemented using the Rete algorithm. (For a detailed analysis of parallelism in production systems, see [Gup86].) We address a different problem that appears superficially similar. In multi-user RBDEs, parallelism is *intrinsic* and not a technique used for improving the performance of the expert system.

In this paper we explore the problems of synchronization and cooperation in multi-agent expert systems applied to the domain of software development. We outline a solution that divides the problem into three components: (1) how to detect potential conflicts between the concurrent activities of developers; (2) how to specify the consistency requirements of a particular project; and (3) how to use the consistency specification to resolve potential conflicts. Our solution exploits recent advances in extended transaction models [BK89], which provide clues as to how to support cooperation while guaranteeing the consistency of data. This research was motivated by our previous work on a single-user RPDE called MARVEL [KFP88, KBFS88, KBS90].

We first describe the basic single-agent RBDE model and show how supporting multiple agents complicates the model. Next we introduce our approach to solving the multi-agent problem. We then sketch how we detect and resolve conflicts between con urrent rule chains based on a rule-based specification of the project's consistency that supports synergistic cooperation among human developers.

2 The RBDE Model

RBDEs provide assistance during the development of a particular software project by automatically firing rules that manipulate the components of the project, each of which is abstracted as an object and stored in the project database. The rules encapsulate software development activities (such as editing, compilation, debugging, etc.), which are performed by the conventional software development tools available outside the RBDE (e.g., the editors, compilers, word processors, etc.). By *encapsulating* an activity in a rule, we mean that the rule specifies the condition that must be satisfied in order for the tool that performs the development activity to be invoked on the specified objects, and the effects of invoking the tool on the objects. Since the development of different projects often requires different sets of tools, and since each project has its own restrictions on how these tools must be used, the rules must be made explicit and provided to the RBDE rather than built into its interpreter. We envision that a project administrator will write the development rules of the project and load them into the RBDE.

The users of an RBDE do not have to be aware of the existence of rules. They will simply

*Barghouti is supported in part by the Center for Telecommunications Research.
†Kaiser is supported by National Science Foundation grants CDA-8920080, CCR-8858029 and CCR-8802741, by grants from AT&T, BNR, Citicorp, DEC, IBM, Siemens, Sun and Xerox, by the Center for Advanced Technology and by the Center for Telecommunications Research.

```
rule:

   condition

   { activity }

   effect1; effect2; . . .
```

Figure 1: A Generic Rule in the RBDE Model

request commands, each of which will either correspond to a rule or be built into the environment (e.g., commands to add and delete objects). If the command corresponds to a rule that causes a chain of other rules to be invoked (explained shortly), the user will see only the results of the chain of activities that the RBDE is automatically performing on his or her behalf. As shown in figure 1, each rule has a condition that must be satisfied before the second part, the development activity the rule encapsulates, is executed. The condition is essentially a query (i.e., read operation) on the status of objects in the database. Due to lack of knowledge about the internals of the tools invoked and the human decisions that will be made during the activity (e.g., for interactive tools), the activity is modeled as a "black box" whose inputs and possible outputs are known, but in order to determine which of several possible outputs it will produce, the black box must be executed. The third part of each rule is a set of mutually exclusive effects, each of which changes (i.e., writes) the status of objects in the database. The results of the rule's activity determine which effect to assert. An example rule understood by our MARVEL system is shown in figure 2. We have intentionally chosen simple rules because we are mainly concerned with how the rules interact through their conditions and effects, and the examples demonstrate that aspect.

If the effect of a rule changes the database in such a way that the conditions of other rules become satisfied, all of those rules are fired automatically. This kind of forward chaining is similar to what is implemented in OPS5 production systems, except that all the rules whose conditions become satisfied are invoked rather than just one chosen by a conflict resolution strategy. Alternatively, if the condition of the original rule matching the user command is not satisfied, backward chaining is performed to attempt to satisfy it. Although this sounds similar to backward chaining in theorem provers, constraint systems and some production systems, there are peculiarities to the RBDE model that are discussed shortly. RBDEs implement a chaining model in order to either maintain consistency, as defined by the conditions and effects of rules, or automate the performance of some activities that would otherwise be done manually.

2.1 Rule Execution Model

We assume a model in which rules define automation, and both forward and backward chaining are supported. This model is implemented in several RBDEs including MARVEL. The main aspect that distinguishes this model from chaining models used in conventional AI systems is the fact that rules may invoke external tools, whose effects can not be determined before the tool invocation terminates, and which might perform some actions that are irreversible as a side effect of running the tool. This fact, which is a requirement of the domain, manifests itself in two ways: (1) the tools operate on the file system outside the project database, and whatever these tools change while they


```
build[?p:PROGRAM]:

    # if all the libraries belonging to the project (instance of the
    # PROJECT class) that contains the program have been archived,
    # and if all the C files (instances of the class CFILE)
    # that are components of the program have been successfully
    # analyzed and compiled, then the program can be built by calling
    # the BUILD tool.  Depending on the result of the tool, update the
    # status attribute of the program to be either Built or NotBuilt.

    (and (forall PROJECT ?P suchthat (member [?P.programs ?p]))
         (forall LIB ?l suchthat (member [?P.libraries ?l]))
         (forall CFILE ?c suchthat (member [?p.cfiles ?c])))
    :
    (and (?c.analyze_status = Analyzed)
         (?c.compile_status = Compiled)
         (?l.archive_status = Archived))

    { BUILD build_program ?p }     # this is the activity of the rule

    (?p.build_status = Built);     # postcondition in case of success
    (?p.build_status = NotBuilt); # postcondition in case of failure
```

Figure 2: Example of a Marvel Rule

are running cannot be monitored by the RBDEs, which only has control over changes done to the project database; and (2) the existence of multiple mutually exclusive effects of a rule. Because of these two aspects, the nature of backward chaining in the RBDE model is different from other rule-based systems.

The difference is that rules fired in a backward chaining cycle may have invoked external tools, whose actions cannot be reversed if the chaining fails in satisfying the condition that triggered it. This is a direct implication of not being able to monitor the changes that an external tool caused on the file system, and thus not being able to fully undo them. For example, suppose that rule r1 invokes a mail tool that sends a message to the manager of a project, and has two effects e1 and e2, while rule r2 has a condition c1, which can be satisfied if e1 is asserted. Assume that a developer requests a command that corresponds to r2, which can not be immediately executed because its condition c1 is not satisfied. In order to try to satisfy this condition, the RBDE might automatically invoke r1. However, it cannot be determined a priori whether invoking r1 will satisfy the condition of r2 because it might happen that the second effect (i.e., e2) is the one that will be asserted rather than e1. In this case, if there is not other rule with an effect that satisfies c1. the backward chaining fails and the rule r2 can not be executed. The mail tool, however, would have already been invoked when r1 was fired, sending a message to the manager, which can not be undone. In most other rule-based systems the effects of a failed backward chaining cycle can be reversed.

2.2 Multi-Agent RBDEs

Most existing RBDEs are single-agent in the sense that they allow only one activity to be performed at any one time on any one database. Some RBDEs require one developer to lock the whole project database, while other RBDEs allow multiple developers to access different copies of the whole database concurrently, forcing them to merge their changes manually later. Merging changes is often a cumbersome task, and is sometimes not doable if the changes conflict with each other. In all single-agent RBDEs, the activities performed on the project database (i.e., any single and consistent copy) are strictly serial. They do not allow activities to execute in the background while the developer executes a foreground activity. In addition, they do not allow two developers to access the same copy of an object in the database, thus preventing cooperation.

When multiple developers cooperate on a project within an RBDE, they share a common database that contains all the objects of the project. Each developer starts a *session* in order to complete his or her assignment; the sessions of multiple developers may execute concurrently and the developers may concurrently request operations that access objects in the shared project database. This presents several new technical challenges, the most serious of which is the interaction among concurrent chains of rules, which might introduce *conflicts* that violate the consistency of the objects they access. More specifically, forward and backward chaining in RBDEs cause the firing of a chain of rules on behalf of an individual developer in such a way that may affect other developers: the rules may modify objects seen by other developers and/or their own rule chains, and may logically negate the conditions or effects of rule chains executing on behalf of other developers.

There is a need to synchronize concurrent operations if they change the same object, or objects that are semantically dependent on each other. Synchronization in an RBDE involves not only the human developers but also the rules that automatically perform operations on their behalf, since those rules also access the shared database. We use the term *agent* to refer to both human developers and rules that are fired on their behalf. What is missing in existing RBDEs is the ability to synchronize concurrent accesses by multiple agents to the shared database while still providing an environment that supports cooperation.

A simple example illustrates how multiple agents complicate automation in RBDEs. Assume that two developers, Bob and Mary, are working on a common chore. Bob requests a command that triggers a forward chaining cycle. Before the forward chaining resulting from Bob's command is finished, Mary requests a command that corresponds to a rule whose condition is not satisfied, and thus initiates a backward chaining cycle in order to satisfy it. One problem is that one of the rules on the chain triggered by Bob's command might conflict with one of the rules on the in-progress chain that Mary has triggered. Alternatively, a conflict might have occurred even in a single-user context if, for example, Bob had requested two commands concurrently (in two different windows). This problem arises when the RBDE allows multiple rule chains to execute concurrently.

The trivial solution of simply serializing chaining cycles is not satisfactory, since the activities that are automatically invoked might take an arbitrary amount of time and it would not be possible to cooperate by sharing knowledge during the chaining. What is needed is a concurrency control mechanism that allows cooperation among multiple agents. This cooperation is *synergistic* in the sense that the total effect of the cooperation among the developers cannot be achieved by isolating them, i.e., forcing them to take turns in developing common parts of the project.

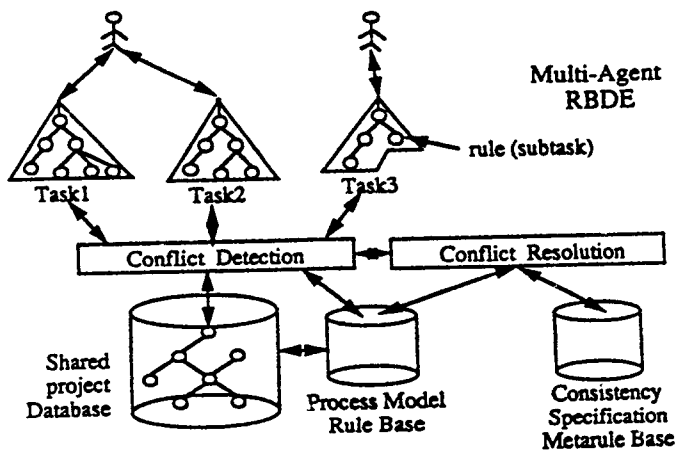


Figure 3: The Multi-Agent Problem in RBDEs

3 Overview of our Approach

The overall behavior of cooperating developers in an RBDE can be modeled as concurrent requests by users to execute environment commands, each of which might cause the firing of a backward rule chain and a forward rule chain; these rule chains then execute concurrently, performing operations on objects in the shared project database. We divide the synchronization problem into three subproblems, which are depicted in figure 3.

- **Conflict detection:** before firing a rule within an in-progress rule chain, the RBDE must decide whether or not the rule could cause access conflicts with any other rule that has already been fired in the same chain or concurrent chains. If it does, a potential conflict is detected and must be resolved before evaluating the rule's condition and invoking it's activity.
- **Consistency specification:** specifying the kind of consistency that must be maintained in the database of the particular project, which might be different for different projects. This specification, together with the rules, represent the domain-specific knowledge that enables us to attack the problem of cooperation.
- **Conflict resolution:** once a potential conflict situation has been detected, the RBDE must resolve it according to the consistency specification. If the conflict situation violates the specified consistency, then it must be prevented or repaired somehow; otherwise, it can either be ignored or some other action, such as warning a user, can be taken.

Our approach to solving these problems is as follows:

1. To solve the conflict detection problem, we encapsulate *rule chains* (i.e., all the rules that are triggered by a user command and the RBDE's consequent backward and forward chaining on the user's behalf) in logical units, called *tasks*, which resemble transactions in database systems. Then, we use a conflict detection protocol based on the concept of serializability to detect potential conflicts between tasks. The protocol uses the collections of objects accessed by concurrent tasks as a basis for detecting conflicts.

2. To solve the consistency specification problem, we have designed a framework in which it is possible to: (1) define conflict situations using our concept of troupes of rules (defined shortly); and (2) prescribe how the conflict situation should be resolved.

3. To resolve a detected conflict, we have developed a synchronization mechanism that uses the consistency framework to achieve a reasonable compromise between two conflicting goals:

   (a) Maximize concurrent access by multiple agents to the shared project database, and thus maximize the number of operations that can proceed concurrently. This has the effect of decreasing the environment's response time to user commands, and thus improves the overall productivity of the team of cooperating developers.

   (b) Maintain the project database in a consistent state as specified in the consistency framework, thus reducing the time wasted on repairing inconsistent data, and thereby facilitating the successful completion of the software project.

4 Detecting Conflicts Between Concurrent Agents

We explain the conflict detection problem by means of an example. Suppose that Bob and Mary want to test module ModA by running a test suite, S. ModA consists of two procedures, p1 and p2. When Bob and Mary request commands concurrently, Bob's command might trigger a chain of rules, one or more of which might conflict with one or more of the rules in the in-progress chain that Mary's command has triggered.

Suppose that Bob requests a command `test modA` (corresponding to the test rule shown in figure 5) at time t1, which triggers the forward chaining cycle shown in figure 4. Mary requests a command `test p1` (corresponding to the second test rule in figure 5) at time t2. The condition of the rule is satisfied at that time, causing the activity of the rule to be invoked. Meanwhile, Bob discovers that p1 has a bug so he issues the command `modify p1` at time t3, which triggers a backward chain to `reserve p1` before calling the editor on p1. The sequence of activities depicted in figure 4 causes a conflict because the effects of the `reserve` rule in Bob's chain will negate the condition of Mary's `test` rule, already in-progress. This causes Mary's test to be invalid since the procedure she is testing is about to be modified by Bob.

To detect conflicts similar to the one presented above, each chaining cycle triggered in response to a developer's command is encapsulated in a *task*. Each rule in the task is a *subtask* of the task representing the whole chain. A potential conflict between concurrent tasks occurs when the firing of a rule (subtask) within one of the tasks violates the serializable execution of the in-progress tasks. An execution is said to be serializable if an equivalent serial execution can be found.

The conflict detection mechanism detects a violation of serializability by using a *nested incremental locking* (NIL) protocol based on the standard two-phase locking (2PL) protocol used in most traditional DBMSs [EGLT76]. 2PL divides each transaction into two phases, a growing phase, in which locks on objects are acquired, and a shrinking phase, in which locks are released. The shrinking phase begins with the first release of a lock, and during that phase the transaction can not acquire a new lock. 2PL ensures that only serializable schedules of transactions are allowed.

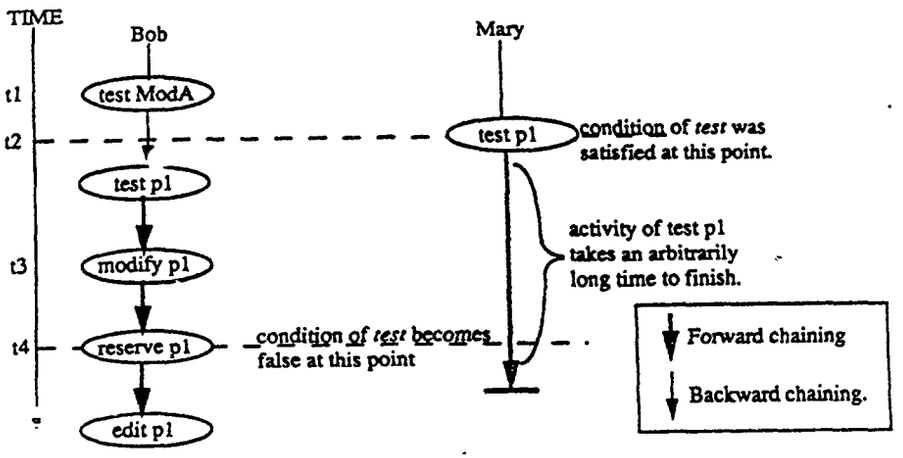


Figure 4: Example of the Conflict Detection Problem

Like 2PL, NIL associates two locks with every object in the database, a read lock and a write lock. These locks are internal to the conflict detection mechanism and invisible to developers and the rest of the system. When initiating a rule and before evaluating the condition of the rule, read locks must be acquired by the parent task on all the objects accessed in the condition, and write locks on all the objects accessed in its effects (these are added to the set of locks previously acquired, hence the protocol is *incremental*). A conflict is detected when a rule attempts to set a read or write lock when another rule already holds the write lock; multiple rules can simultaneously hold the read lock in any case, but multiple rules can simultaneously hold the write lock only if this is permitted by the conflict resolution mechanism, described later. A conflict is detected even when the two rules are part of the same task, thus the protocol is *nested*.

The objects accessed by the condition of a rule are determined by the logical clause of the condition, which we call the *characteristic function*. This function describes a *collection* of objects, each of which possess the property that characterizes the collection. The collection is thus a dynamically defined set whose members change when the values of their attributes change. An example of a collection is the set of all objects that are both instances of the `PROCEDURE` class and contained in module `ModA`. This collection is used in the `test` rule of figure 5. The effects of a rule only write the values of the attributes of objects passed to the rule as actual parameters.

Like 2PL, NIL requires that all tasks be *well-formed* in order to test for serializability: (1) they do not relock objects that have been locked earlier in the task, and (2) they are divided into a growing phase, in which locks are only acquired and a shrinking phase, in which locks are only released. During the shrinking phase, a task is prohibited from acquiring locks. This is done by acquiring locks as the rule chain grows, and not releasing any locks until the chain terminates.

If a rule tries to acquire a lock that has already been acquired by a rule that is a subtask

```
(rule test (MODULE ?mod; TEST ?test)
  condition: (forall PROCEDURE ?p suchthat (contains ?mod ?p)
               (?p.test-status = tested))
  activity: (run ?test ?mod)
  effects: (xor (?mod.test-status = tested)
                (?mod.test-status = failed))
  comment: ''We can test a module only after all of its component procedures
            have been tested.  'Run' is the name of the tool that
            executes the test bound to the variable ?test on the module
            bound to ?mod. If the module passes the test, set its
            test-status attribute to 'tested'; otherwise, to 'failed'.'')

(rule test (PROCEDURE ?proc; TEST ?test)
  condition: (?proc.availability = available)
  activity: (run ?test ?proc)
  effects: (xor (?proc.test-status = tested)
                (?proc.test-status = failed))
  comment:''A procedure can be tested only if it is available; i.e.,
           it is not reserved for editing by any user.'')

(rule edit (PROCEDURE ?proc; USER ?user)
  condition: (and (?proc.availability = reserved)(?proc.reserver = ?user))
  activity: (edit ?proc)
  effects: (?proc.status = changed)
  comments: ''We can edit a procedure only if it has been reserved by
             the same user who initiated the edit rule.'')

(rule reserve (OBJECT ?obj; USER ?user)
  condition: (?obj.availability = available)
  activity: (reserve ?obj ?user)
  effects: (and (?obj.availability = reserved)(?obj.reserver = ?user))
  comments: ''A user can reserve an object if it is available in the
             sense that no other user has reserved it.'')
```

Figure 5: Example Rules Based on Marvel Notation

of another task, this is a *conflict*. The rule does not wait for a lock when there is a conflict, as might be done in a conventional database management system, but instead the conflict resolution mechanism is invoked; thus there is no possibility of deadlock, where one task would be waiting for another task to release a lock while this other task is waiting for the first task to release another lock.

In the example of figure 4, when Bob requests the `test ModA` command, the RBDE initiates a task T_{Bob} and fires the corresponding `test` rule as a subtask. Before the activity of the rule is invoked, T_{Bob} acquires a read lock on the test suite S and on all the procedures contained in ModA (i.e., p1 and p2) and a write lock on the object representing ModA. Similarly, when Mary requests the command `test p1`, T_{Mary} acquires a read lock on p1. Now when Bob's task fires `modify p1`, it tries to acquire a write lock on p1, but discovers that p1 has already been locked by T_{Mary} in an incompatible mode, thus causing a conflict.

Once detected, conflicts must be resolved by consulting the conflict resolution mechanism, which uses the consistency specification of the particular project to decide on how to resolve the conflict. In the example, it might be allowable for Bob and Mary to edit and test p1 at the same time because they are cooperating together closely, and Mary knows that she is testing a feature of p1 that Bob's modification will not change; her test would still be semantically valid in this case, but the special knowledge is needed for the RBDE to determine this.

5 Conflict Resolution

Unlike in traditional database management, detecting violations to serializable access to objects does not automatically mean that the consistency of the database will be violated, as demonstrated by the previous example. The conflict detection mechanism presented above detects all potential conflicts (i.e., non-serializable access to objects) and leaves it to the conflict resolution mechanism to consult the specific consistency requirements of the project and decide if any of them are actually violated by the detected conflict.

5.1 Troupes of Rules for Consistency Specification

In the domain of software development, non-serializable access to data and cooperation between concurrent tasks are required in many cases. The situations in which such non-serializable access violates the consistency of the project depends on the database operations implied by the rules in the particular project. The rules for different projects are different and thus the RBDE must be provided with knowledge about allowable interactions between rules in a specific project. The problem is finding a framework for specifying which interleavings between concurrent tasks are allowable (i.e., do not violate consistency). The framework should specify the *granularity* at which different database operations (performed to either evaluate the condition of a rule or assert its effects) can be interleaved. This specification framework can then be used by the conflict resolution algorithm to provide maximum concurrency while maintaining consistency.

We define project-specific consistency in terms of *control rules* that operate on the process model rule base. Each control rule has a condition that describes a conflict situation, and a repair that prescribes how to resolve the specific conflict. The conflict situation is described using two concepts: collections of objects and troupes of rules. Using collections of objects, a control rule can refer to all members of a teams of developers (each developer is an object and a team is a collection of objects that shares the same responsibility), all objects that are accessed by a task, etc.

Troupes define sets of rules that share a particular semantic property; these sets are specified in advance (by the project administrator who writes the rules describing the development process

```
COMMUTATIVITY-TROUPE:
    r1, r2, r3, r4;
    r1, r5, r6;
    . . .
```

Figure 6: Example of a Troupe of Rules

of the project), along with the data model and process model. Each set is considered an instance of the troupe in which it is defined. Troupes are like classes in that they prescribe methods, or actually one specific method, in the form of a control rule. It might be interesting to further extend the concept of class to troupes by introducing attributes of troupes and inheritance among troupes, but here we consider only troupes that define maximal sets of rules sharing a mutual property. For example, the COMMUTATIVITY-TROUPE defined in figure 6 defines several sets of rules that can all commute with each other (the knowledge is to which rules commute is provided by the project administrator). In other words, it does not matter in which order we run the rules that are members of an instance of this troupe. Both collections of objects and troupes of rules provide semantic knowledge that can be used by control rules to resolve conflicts. Figure 7 gives two control rules. The notation in the figure is not the actual syntax of any particular RBDE language but is intended to demonstrate the idea of project-specific consistency constraints.

5.2 Consistency-Based Conflict Resolution

The conflict resolution mechanism uses the control rule base to decide what to do with a conflict passed to it by the conflict detection mechanism. For example, the conflict discovered in the example of figure 4 matches the IF part of the lock-conflict control rule of figure 7. In this case, the conflict resolution mechanism would check whether Mary and Bob are members of the same user group, whose members cooperate closely, and if they are, it will ignore the conflict; otherwise, the mechanism would reject Mary's request to test p1. The second control rule in figure 7 can be used to resolve a conflict that occurs between a rule just being initiated by a user and the in-progress rule in another task initiated by another user. The conflict situation of the control rule depicts this situation, while the repair part checks if the two rules belong to any instance of the COMMUTATIVITY-TROUPE, i.e., if the order of execution of the rules does not affect the results of the tasks.

6 Conclusions

Our approach to multi-agent expert systems is specific to our domain, rule-based development environments, where the primary challenge is to find a mechanism that supports both synchronization to ensure consistency and cooperation to permit concerted efforts. We defined the task, or rule chain, as the basic unit of concurrency control. We detect potential conflicts between concurrent tasks using a general purpose protocol, but then exploit domain-specific knowledge to determine which of these potential conflicts violate the consistency of the shared project database. Potential conflicts that do not violate consistency permit cooperation among the tasks.


```
(ControlRule lock_conflict
  IF  (and (in-progress TASK ?t1 USER ?u1)
           (in-progress TASK ?t2 USER ?u2)
           (has-lock ?t1 ?obj ?lock_mode_1)
           (request-lock ?t2 ?obj ?lock_mode_2)
           (not-compatible ?lock_mode_1 ?lock_mode_2))
  THEN
      (if (and (exists USER_GROUP ?g)
               (contains ?g ?u1)
               (contains ?g ?u2))
       then
          (grant-lock ?t2 ?obj ?lock_mode_2
           ''Warning: user ?user2 also has ?obj locked'')
       else
          (reject-lock ?t2 ?obj
           ''Conflict: user ?user2 has ?obj locked''))

  COMMENT: ''Two users who are members of the same developer team
            (where USER_GROUP is a collection of objects, each of
            which represents a user) can share an object, while other
            users who are not members of the same team will not be
            granted permission to lock the object.'')

(ControlRule test_conflict
  IF  (and (in-progress RULE ?r1 USER ?u1 TASK ?t1)
           (start RULE ?r2 USER ?u2 TASK ?t2)
           (not-equal ?t1 ?t2)
           (not-equal ?u1 ?u2)
           (conflict ?r1 ?r2))
  THEN
      (if (exists COMMUTATIVITY-TROUPE ?t such_that
                      (and (member ?r1 ?t)
                           (member ?r2 ?t)))
       then (ignore-conflict ?r1 ?r2)
       else (abort ?r2))

  COMMENT: ''If the activity of the rule whose initiation caused the
            detection of conflict can commute with the in-progress
            rule with which it conflicted, then ignore the potential
            conflict since execution order does not matter.
            Otherwise, abort the later rule.'')
```

Figure 7: Examples of Consistency Control Rules

Acknowledgments

We would like to thank Michael Sokolsky, who collaborated with us on designing and implementing MARVEL; Israel Ben-Shaul, who is working on the multi-user object management system for MARVEL; and George Heineman, who is working with us on developing an RBDE model that combines consistency preservation and automation. The single-user MARVEL version 2.6 is available for licensing to educational institutions and industrial sponsors; contact Israel Ben-Shaul, israel@cs.columbia.edu, 212-854-2930 for information.

References

[BK89] N. S. Barghouti and G. E. Kaiser. Concurrency control in advanced database applications. Technical Report CUCS-425-89, Columbia University Department of Computer Science, New York, NY, November 1989. Submitted for Publication.

[CLF88] University of Southern California, Information Sciences Institute, Marina del Rey CA. *CLF Manual*, January 1988

[EGLT76] K. Eswaran, J. Gray, R. Lorie, and I. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11) 624–632, November 1976.

[Gup86] A. Gupta. *Parallelism in Production Systems*. PhD thesis, Carnegie Mellon University, Department of Computer Science, March 1986. Technical Report CMU-CS-86-122.

[HL88] K. E. Huff and V. R. Lesser. A plan-based intelligent assistant that supports the software development process. In *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 97–106, Boston, MA, November 1988. ACM Press. Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.

[KBFS88] G. E. Kaiser, N. S. Barghouti, P. H. Feiler, and R. W. Schwanke. Database support for knowledge-based engineering environments. *IEEE Expert*, 3(2):18–32, Summer 1988.

[KBS90] G. E. Kaiser, N. S. Barghouti, and M. H. Sokolsky. Preliminary experience with process modeling in the marvel software development environment kernel. In *23rd Annual Hawaii International Conference on System Sciences*, volume II, pages 131–140, Kona HI, January 1990.

[KFP88] G. E. Kaiser, P. H. Feiler, and S. S. Popovich. Intelligent assistance for software development and maintenance. *IEEE Software*, 5(3):40–49, May 1988.

[MR88] N. H. Minsky and D. Rozenshtein. A software development environment for law-governed systems. In *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 65–75, Boston MA, November 1988. ACM Press. Special issue of *SIGPLAN Notices*, 24(2), February 1989 and of *Software Engineering Notes*, 13(5), November 1988.

[Nii86] H. P. Nii. Blackboard systems. Technical Report STAN-cs-86-1123, Department of Computer Science Stanford University, June 1986. Appears in AI Magazine, Volumes 7-2 and 7-3.

[Per89] D. Perry editor. *5th International Software Process Workshop*, Kennebunkport ME, October 1989. ACM Press. In press.

SYNCHRONIZATION PROBLEMS OF COMPENSATE OPERATIONS IN THE OBJECT-MODEL

Makoto TAKIZAWA*) and S. Misbah DEEN+)

*)Dept. of Information and Systems Engineering
Tokyo Denki University
Ishizaka, Hatoyama, Hiki, Saitama 350-03, Japan
Tel. +81-492-96-2911 ext.2406 Fax. +81-492-96-6185(or 0501)
e-mail taki@takilab.k.dendai.ac.jp

+)DAKE Centre and Dept. of Computer Science
University of Keele
Keele, Staffordshire, ST5 5BG, England
Tel. +44-782-621111 ext.3354 Fax. +44-782-713082
e-mail misbah@cs.kl.ac.uk

ABSTRACT

In order to integrate database systems and knowledge base systems, it must be discussed how to keep the systems consistent on the presence of various failure. One conventional way to recover the system is to restore the old state stored in the log. However, in action-oriented systems like air- traffic control systems and process control systems, once some action like a launch of missile is taken, we cannot restore the old state. The conventional database systems provide some recovery mechanism by which the systems restore the old states by using the states saved in the log or back up copies. However, after some operation is executed, if some trouble like system failure is found, the system state must be transited to some consistent state which may be different from the old state. In this paper, we propose one method where some operations named *compensate* operations are executed. Before executing operations on objects, the objects have to be locked in order to synchronize the interleaved or parallel executions of operations from multiple transactions. The compensate operations also require the locks on the objects. In this paper, we discuss not only what is the compensate operation but also the relation among the compensate operations and lock modes.

1. INTRODUCTION

In order to integrate database systems and knowledge base systems, it must be discussed how to keep the systems consistent on the presence of various failure. One conventional way to recover the system is to restore the old state stored in the log. In action-oriented systems like air-traffic control systems and process control systems, once some action is taken, we cannot go back to the old state. For example, after some missile is launched, we cannot go back to the old state in which the missile is not launched. However, conventional database systems provide only recovery mechanisms which can restore the old state by using the state log and backup copies [BERN87, HAER83]. Even after some operation which cannot be recovered by the state log is executed, if some trouble like failure is found, the system state must be transited to some consistent state. In order to do that, the system provides some operations which can be used to transit to some consistent state. Such operations are named *compensate* operations.

A transaction is an atomic unit of work and is composed of multiple operations. Each operation can call operations on another objects. In this sense, transactions are tree-structured and each operation has to be executed as a transaction, i.e. *nested* [MOSS85, TRAI83, LYNC86, WEIH89, BEER89]. In this paper, locking mechanism [ESWA76] is considered as the synchronization method since it is used widely. Before executing operations on objects, the objects have to be locked in order to synchronize the interleaved or parallel executions of operations from multiple transactions. There are two synchronization methods of transactions, i.e. *close-nested* [MOSS85] and *open-nested* [GRAY81, TRAI83] ones. In the open-nested execution, each operation *a* releases locks held by the operations called by *a* when *a* commits. On the other hand, in a close-nested case, each operation *a* does not release locks even if *a* commits. All the locks are released only when the root commits. Open-nested transactions can imply more concurrency than close-nested transactions. In [TAKI90], a *semi-open-nested* execution scheme of transactions is presented, which is a mixture of both schemes.

The compensate operations also require the locks on the objects. As stated in [TAKI90], some *unresolvable* deadlock which cannot be resolved by executing compensate operations might occur. Also, [TAKI90] shows that no unresolvable deadlock occurs in some kind of system named *minimum acyclic* system under an assumption that every compensate operation can use the object when the operation uses it. However, neither the meaning of the locking mechanism nor the relation among the locking model and the compensate operations is clear in [TAKI90]. In this paper, we discuss the locking model and the general relation among the compensate operations and lock modes.

In section 2, we give definitions of transactions. In section 3 we discuss what compensate operations are. In section 4, we discuss the locking mechanism for multiple lock modes. In section 5, the synchronization mechanism for executing the compensate operations is discussed. In section 6, we discuss the deadlock problems caused by interleaved execution of compensate operations. In section 7, we discuss how to find more efficient operation sequences to undo transactions.

2. OBJECT MODEL AND TRANSACTIONS

A system M is composed of a set OO of objects and a set PP of operations. Each object o in M provides some data structure and a set of operations on the data structure. Let P_o be a set of operations provided by o. An operation a in P_o can call an operation b provided by another object. A unit of work is a transaction which is composed of multiple operations. A *transaction* T is represented in an ordered tree named a transaction tree, where each node denotes an operation execution and each parent-child relation indicates a caller-callee one. Leaf operations in T are named *actions* and non-leaf operations *subtransactions*. Each action manipulates directly the object. Each subtransaction for an object o can call actions on o and another subtransactions. *[[a]]* is used to explicitly denote that a is an action. In this paper, transactions are assumed to be executed serially.

[Example 2.1] Fig.1 shows an example of a transaction. T is a root which calls an operation a. a, b, c, e, and g are subtransactions, and d, h, f, i, and j are actions. A subtransaction b on an object o_b first calls another subtransaction c which manipulates another object o_c, an action h which manipulates the same object o_c as c, and then d is executed. The operations are executed in a sequence $a, b, c, h, d, e, f, g, i, j$.

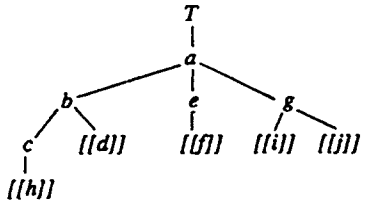


Fig.1 Transaction tree T

Here, let lower-case italic letters like a, b, ... denote operations, and upper-case italic letters like A, B, ... denote sequences of operations. For sequences A and B, $<A; B>$ denotes a concatenation of them. Let λ be an empty sequence. Let $A|_a$ be a postfix of A whose top is a, $A|^a$ be a prefix of A whose last is a, and $A|_a^b$ be an infix of A whose top is a and last is b. For example, for $A = <a; b; c; d; e>$, $A|_c$ is $<c; d; e>$, $A|^b$ is $<a; b>$, and $A|_b^d$ is $<b; c; d>$. Here, for each operation a in T, let *[a* and *a]* denote a *begin* and *end* of a, respectively. For each operation a in a transaction T, the i- th *expansion* a^i of a is defined as follows.

[Definition] (1) $a^i = [[a]]$ if a is an action, (2) $a^i = a$ if $i = 0$ or a is a begin/end, and (3) $a^i = <[a; a_1^{i-1}; ...; a_n^{i-1}; a]>$ if a calls operations a_1,

Since T is a finite tree, there exists some fixed point such that $a^i = a^j$ for $j > i$. Let a^* denote the fixed point named the *full expansion* of a. T^* denotes a sequence of executed operations of T. We extend the expansion to sequences. A^i is said to be an *i-level expansion* of a sequence A *iff* (1) $A^i = a^i$ if A is an operation a, (2) $A^i = A$ if A is a begin/end, and (3) $A^i = <a^i; A_2^i>$ if $A = <a; A_2>$. The *full expansion* A^* is defined in the same way as a^*. A^{-i} and A^{-*} are *i-level reduction* and *full reduction* of A, which are defined to be inverses of A^i and A^*, respectively.

[Example 2.2] In Example 2.1, $T^1 = <[T; a; T]>$ which means that T calls a. $T^2 = <[T; a^1; T]> = <[T; [a; b; e; g; a]; T]>$ which denotes more detailed description of T, i.e. a calls operations b, e, and g. $T^3 = <[T; [a; b^1; e^1; g^1; a]; T]> = <[T; [a; [b; c; [[d]]; b]; [e; [[f]]; e]; [g; [[i]]; [[j]]; g]; a]; T]>$. Finally, $T^4 = T^5 = ... = T^* = <[T; [a; [b; [c; [[h]]; c]; [[d]]; b]; [e; [[f]]; e]; [g; [[i]]; [[j]]; g]; a]; T]>$. T^* denotes a most detailed execution sequence of operations of T. On the other hand, $<[a; b; [e; [[f]]; e]; g; a]>^{-1} = <[a; b; e; g; a]>$, and $<[a; b; [e; [[f]]; e]; g; a]>^{-*} = <a>$. □

[Definition] A is said to be *closed iff* there exists some operation a such that $a^* = A^*$. A is said to be *basic iff* $A = A^*$. A and B are said to be *equivalent* ($A \Leftrightarrow B$) *iff* $A^* = B^*$. □

[Definition] A is said to be greater than B ($A >> B$) *iff* (1) for some i, $A = B^i$, or (2) there are some A_1, A_2, B_1, B_2 such that $A = <A_1; A_2>$, $B = <B_1; B_2>$, $A_1 >> B_1$, and $A_2 >> B_2$. □

A closed sequence is in a form $<[a; ...; a]>$. A basic sequence includes only actions and begin/ends. T^* is a closed and basic sequence which denotes an execution of T. In Example 2.2, T^2 is closed but not basic. It is clear that for sequences A and B, $A^* = B^*$ *iff* $A^{-*} = B^{-*}$, and $A \Leftrightarrow B$ if $A << B$. Let $[A]$ denote a set of sequences equivalent to A, i.e. $\{ B \mid A \Leftrightarrow B \}$.

[Example 2.3] In Fig.1, $a^1 = <[a; b; e; g; a]>$ and $a^2 = <[a; [b; c; [[d]]; b]; [e; [[f]]; e]; [g; [[i]]; [[j]]; g]; a]>$. $a^* = <[a; [b; [c; [[h]]; c]; [[d]]; b]; [e; [[f]]; e]; [g; [[i]]; [[j]]; g]; a]>$. Let A be $<[a; b; [e; [[f]]; e]; g; a]>$ and B be $<[a; [b; c; [[d]]; b]; e; g; a]>$. a, a^1, a^2, a^*, A, and B are equivalent. $a << a^1 << a^2 << a^*$. $a^1 << A << a^2$ and $a^1 << B << a^2$, but neither $A << B$ nor $B << A$. □

The system M changes the states each time when an operation is executed. Here, let SS be a set of possible states of M. For every operation a, let $a(s)$ denote a state obtained by applying a on a state s in SS. Let T^s denote a sequence of executed operations of T in a state S. Let $C(T^s)$ be a current operation which is being executed in S. For example, suppose that T is executed until *[c* in a state S in Fig.1. $T^s = <[T; [a; [b; [c>$ where $C(T^s) = [c$.

We write $a \rightarrow_c b$ if a calls b. $\rightarrow_c^*$ is a transitive closure of $\rightarrow_c$. $a \Leftrightarrow_s b$ if a and b are operations provided by some object. $a \rightarrow_s b$ if (1) $a \rightarrow_c b$, or (2) there is some operation c such that $a \rightarrow_c c \Leftrightarrow_s b$ or $a \Leftrightarrow_s c \rightarrow_c b$. $\rightarrow_s^*$ is a transitive closure of $\rightarrow_s$.

[Definition] A system M is said to be *cyclic iff* for some operation a, $a \rightarrow_s^* a$. M is said to be *acyclic iff* M is not cyclic. M is said to be *minimum acyclic iff* M is acyclic and for every operation a, b, and c, if $a \rightarrow_s^* b \rightarrow_s c$, then not $a \rightarrow_s c$. □

[Example 2.4] Let us consider a system M which includes file, record, and page objects. Let f be a file operation, r_1 and r_2 be record operations, and p_1, p_2, and p_3 be page operations.
(1) If M is *acyclic*, file-level operations neither can call record-level nor page-level operations, but page-level operations neither can call record- level nor page-level operations. For example,

f calls r_1 and p_1, r_2 calls p_2 and p_3, and r_1 and r_2, p_1 and p_2 are provided by the same object, respectively. That is, since $f \rightarrow_C \{r_1, p_1\}$, $r_2 \rightarrow_C \{p_2, p_3\}$, and $r_1 \Leftrightarrow r_2$, $p_1 \Leftrightarrow p_2$, $f \rightarrow_S r_1 \rightarrow_S p_2$ and $f \rightarrow_S p_2$ [Fig.2].
(2) If M is *minimum acyclic*, file-level operations can call record-level operations but cannot call page-level ones. For example, f calls r_1, and r_2 calls p_1 and p_2. That is, $f \rightarrow_S r_2 \rightarrow_S \{p_1, p_2\}$ but not $f \rightarrow_S \{p_1, p_2\}$.
(3) In the *cyclic* system, each operation can call any operation in M. □

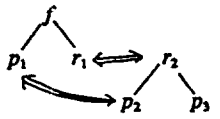


Fig.2 Acyclic System

3. COMPENSATE OPERATIONS

In order to abort a committed operation a, we consider a method of how to execute the compensate operation of a in this paper. Here, we define what is the compensate operation.

3.1. Semantically Equivalent States

[Example 3.1] Let us consider a *B-tree* system B. Let s be some state of B. Let t be a state obtained by applying an *append* operation of a key k to s. Let s' be a state obtained by applying a *delete* operation of k to t. Suppose that some node splits when k is appended. Here, it is clear that s and s' have the same logical structure but they are not identical. From the user's point of view, s and s' are *equivalent*. □

Let SS be a set of states in the system M. That is, some equivalent relation E on SS is given by the application. For every pair of states s and t in SS, s is said to be *semantically equivalent* (or simply *equivalent*) to t on E ($s \sim_E t$) *iff* $\langle s, t \rangle \in E$. There exists the following axiom on E.

[Axiom] For every a in P_a and every s and t in SS, $s \sim_E t$ *iff* $a(s) \sim_E a(t)$. □

That is, let s' and t' be states obtained by applying an operation a on two *equivalent* states s and t, respectively. The axiom says that s' and t' be *equivalent*. In Example 3.1, s and s' are *equivalent*. *delete*(s) and *delete*(s') are *equivalent*.

3.2. Compensate Operations

[Definition] An operation a is said to be *equivalent* to b on E ($a \sim_E b$) *iff* for every state s in SS, $a(s) \sim_E b(s)$. a is said to be *identical* to b ($a = b$) *iff* for every state s in SS, $a(s) = b(s)$. □

[Definition] b is said to be a *compensate* operation of a on E ($b \Rightarrow_E a$) *iff* for every state s in SS, $b(a(s)) \sim_E s$ ($\Rightarrow_E$ is not transitive). b is said to be an *inverse* of a ($b \Rightarrow_I a$) *iff* for every state s, $b(a(s)) = s$ ($\Rightarrow_I$ is not transitive). □

Let $C_E(a)$ be a set of compensate operations of a, i.e. $\{ b \mid b \Rightarrow_E a \}$. Let a^- denote some compensate operation of a, i.e. $a^- \in C_E(a)$. a is said to be *compensatable* on E *iff* a has some compensate operation on E, i.e. $| C_E(a) | > 0$. a is said to be *uncompensatable* on E *iff* $| C_E(a) | = 0$.

[Example 3.2] In Example 3.1, *delete* is a *compensate* operation of *append*, since for every state s in the *B-tree* system B, *append*(*delete*(s)) $\sim_E s$. That is, *delete* $\Rightarrow_E$ *append*. By executing *append*, *delete* can be compensated. As stated in Example 3.1, the state obtained by executing *append* after *delete* may not be identical to the old one. □

[Example 3.3] Suppose that one person A has three bank accounts x, y, and z, and a person B has w. A *transfers* an amount S of money to B's w, i.e. S_x from x and S_y from y ($S = S_x + S_y$). Let us consider the compensate operation *refund* of *transfer*. One method $refund_1$ is to send back S_x to x and S_y to y from w. If A is only interested in his total money in x, y, and z, there is another $refund_2$ where S is transferred from w to z. Both $refund_1$ and $refund_2$ are compensate operations of *transfer*. $refund_1$ is the inverse. □

[Example 3.4] An operation *print-out* does not have the *inverse* operation. However, in real applications like office information systems, if papers would be found to be meaningless after they are printed out, they are thrown away to a dustbin. In this sense, *throw-away* is a *compensate* operation of *print-out*. □

The conventional database systems provide logging mechanisms and backup copies. When some operation a commits, the old state is saved into the log. If the system suffers from some failure, the system restores the old state by reading it from the log. This mechanism provides the *inverse*. If a is a one-to-one mapping, there exists an inverse b of a from the mathematical sense. If not, the inverse of a cannot be defined. However, the state identity constraint is not required in the *compensate* operation. For example, *print-out* does not have the *inverse* operation in Example 3.4 since we cannot go back to the state before some papers are printed out. In this paper, we make the following assumptions.

[Assumptions] (1) Every operation is compensatable.
(2) For every action a on an object o, every compensate of a is an action on o. □

3.3 Compensate Sequence

There is a case that some failure is detected after some sequence A of operations is executed on a state s. In this case, by executing some operation sequence B, the state must be changed to a state equivalent to s. B is said to be a *compensate* sequence of A. For example, suppose that a key k is appended and k is modified to h in the B-tree state s, and then some failure is detected. One method is to execute *modify*$^-$ and *append*$^-$. Another one is to execute *delete* of h. The later depends on the history of operations, i.e. what was done so far. Since it is not easy to think about it, we do not think about the compensate sequence which is obtained by considering the history of operations.

[Definition] A *compensate* A^- of a sequence A is defined as follows

(1) $A^- = a]$ if A is $[a$, (2) $A^- = [a$ if A is $a]$,
(3) $A^- = a^-$ if A is a compensatable operation a, and (4) $A^- = <B^-; a^->$ if A is $<a; B>$. □

[Example 3.5] Let L be a sequence $<[a; b; [c; d; e; c]; a]>$. $L^- = <[a; b; [c; d; e; c]; a]>^- = <[a; [c; e^-; d^-; c]; b^-; a]>$. Suppose that e^- calls operations f and g, and the other operations d and b are actions. $(L^-)^1 = <[a; [c; (e^-)^1; (d^-)^1; c]; (b^-)^1; a]> = <[a; [c; [e^-; f; g; e^-]; [[d]^-; c]; [[b]^-; a]>$. It is noted that e^- is a transaction. □

[Proposition 3.1] Every operation a is compensatable if every action is compensatable.
[Proof] The full expansion a^* includes only actions and begin/ends. Since every action has the compensate, $(a^*)^-$ exists. □

[Definition] A *compensated* sequence $A/$ of a sequence A is defined as follows.
(1) $A/ = a$ if A is an operation a,
(2) $A/ = \lambda$ if $A = <A_1; A_2>$ and $A_1^- \Leftrightarrow A_2$,
(3) $A/ = <A_1; A_4>/$ if $A = <A_1; A_2; A_3; A_4>$ and $<A_2; A_3>/ = \lambda$, and
(4) $A/ = A$ otherwise. □

[Example 3.6] $<a; b; b^-; [c>/ = <a; [c>$. $<a; b; c; d; <c; d>^-; [b; d^-; c^-; b]>/ = <a>$ if $b^1 = <[b; c; d; b]>$. Also, in the B-tree, $<$*append* of k; *delete* of $k>/ = \lambda$ since *delete* is a compensate of *append*. □

[Theorem 3.2] For every sequence A, $A/ \dashv\!\!\!-_E A$.
[Proof] Let A_1 and A_2 be sequences and A_3 be a compensate of A_2, i.e. $<A_2; A_3>/ = \lambda$. For every state s, $A_3(A_2(s)) \dashv\!\!\!-_E s$ from the definition. $A_1(s) \dashv\!\!\!-_E A_3(A_2(A_1(s)))$. Hence, $<A_1; A_2; A_3>/ = A_1 \dashv\!\!\!-_E <A_1; A_2; A_3>$. □

That is, by executing A_3, the effect of A_2 can be compensated. In Example 3.6, $<a; b; b^-; [c>/ \dashv\!\!\!-_E <a, [c>$.

4. LOCKING MECHANISM

4.1 Locking System

Before executing each operation a in P_o on an object o, a must lock o in a mode m_a. A compatibility relation C_o [KORT83] is defined on the lock modes. For every lock mode m and n, $<m, n>$ in C_o *iff* an operation a of a lock mode m can use o which has been used by b of n. Here, for two operations a and b in P_o, $a \Rightarrow_M b$ (a is said to be *compatible* with b) *iff* $<m_a, m_b> \in C_o$. a is said to be *compatible* with a set P of operations *iff* P is empty or for every operation b in P, $a \Rightarrow_M b$. For example, *read* $\Rightarrow_M$ *read*, i.e. even if some *read* operation uses an object o in m_{read}, another *read* can use o, i.e. $<m_{read}, m_{read}> \in C_o$. Let $CM_o(m)$ be a set of modes compatible with a mode m, i.e. $\{ n \mid <n, m> \in C_o \}$, and $IC_o(m)$ be a set of modes incompatible with m, i.e. $M_o - CM_o$.

A locking system L changes a state each time when it accepts an operation. A state is a collection of object states $\{ LS(o) \mid o \in OO \}$. An object state $LS(o)$ is a triplet $(Op(o), hp(o), Wp(o))$ where $Op(o)$ and $Wp(o)$ are disjoint sets of operations ($\subseteq P_o$) and $hp(o) \in Op(o)$. Let $Op(o)$ denote a set of operations which use an object o. Let $h_o(o)$ be an operation which holds a lock on o. Let $Wp(o)$ denote a set of operations which wait on o. Clearly, $Op(o)$ and $Wp(o)$ are disjoint. $top(LS(o))$ is a set $\{ a \mid$ for every b in $Op(o)$ $b \Rightarrow_M a \}$. That is, an operation in $top(LS(o))$ could be a new holder of o when the holder of o releases o.

[Definition] $LS(o) = (Op(o), hp(o), Wp(o))$ is said to be *consistent iff* for every q ($\neq p$) in $Op(o)$, $q \Rightarrow_M hp(o)$. $LS(o)$ is said to be *safe iff* (1) it is consistent, and (2) for every q and r in $Op(o)$, $r \Rightarrow_M q$ or $q \Rightarrow_M r$. A locking system L is said to be *safe iff* every state in L is safe. □

When an operation a requests a lock on an object o or releases o in a state $LS(o) = (Op(o), hp(o), Wp(o))$, $LS(o)$ is changed to another state $LS'(o) = (Op'(o), hp'(o), Wp'(o))$ by the following computation model.

[Model (*LM*) for Locking (a requests o)] (1) If $Op(o) = \phi$, then $hp'(o) = a$, $Op'(o) = \{a\}$, and a is said to be a *holder* of o. Otherwise, if a is compatible with $Op(o)$ ($\cup Wp(o)$), then a can use o, i.e. $Op'(o) = Op(o) \cup \{a\}$, and a is said to be a *user* of o.
(2) Otherwise, a has to wait on o, i.e. $Wp'(o) = Wp(o) \cup \{a\}$. □

[Model (*RM*) for Releasing (a releases o)] (1) If $hp(o) \neq a$, then $Op'(o) = Op(o) - \{a\}$ and for some b in $Wp(o)$ and every operation c in $Op(o)$, if $b \Rightarrow_M c$, then $Op'(o) = Op(o) \cup \{b\}$ and $Wp'(o) = Wp(o) - \{b\}$.
(2) If $hp(o) = a$, $Op'(o) = Op(o) - \{a\}$ and for one operation c in $top(LS(o))$, $hp'(o) = c$. □

It is a scheduling problem which operation b is selected from $Wp(o)$ in (1) and which operation c is selected from $top(LS(o))$ in (2). One method is to select an operation in a chronological order as widely used in the conventional systems. Also, in order to achieve the fairness in the scheduling, even if a is compatible with $Op(o)$, a waits if a is not compatible with $Wp(o)$.

[Example 4.2] Suppose that $Op(o) = \{a, b, c, d\}$ and $\{b, c, d\} \Rightarrow_M a$, $\{c, d\} \Rightarrow_M b$, $d \Rightarrow_M c$. This means that o is requested by a, b, c, and d in this order. Suppose that $a \Rightarrow_M b$. Here, $top(LS(o)) = \{a, b\}$. We can select either one in $\{a, b\}$ as a new holder. It is clear that the state $LS(o) = (Op(o), a$ (or $b), Wp(o))$ is safe. Now, suppose that c is selected as a new holder. In this case, since neither $a \Rightarrow_M c$ nor $b \Rightarrow_M c$ hold, the system is not consistent. That is, a and b are not compatible with the holder c. In the conventional system, a is selected since a is the first operation. It is clear the conventional system is safe because $a \Rightarrow_M b$ if b is older than a. □

There is another locking model to keep the locking system L safe. For an operation a not in P, let $IO(P, a)$ be $\{ b \in P \mid$ not $a \Rightarrow_M b \}$, i.e. a set of operations with which a is not compatible. Let $TO(P, a)$ be $\{ c \in IO(P, a) \mid$ there is no b in $IO(P, a)$ such that $b \Rightarrow_M c \}$.

[Definition] An operation a is said to be *generally compatible* with a set P of operations *iff* (1) a is compatible with P, or (2) for every c in P and every b in $TO(P, a)$, (2-1) $b \Rightarrow_M a$, and

(2-2) if $b \Rightarrow_M c$, then $a \Rightarrow_M c$, and if $c \Rightarrow_M b$, then $c \Rightarrow_M a$. □

[Extended Model (*EM*) for Locking (*a* requests *o*)] If *a* is generally compatible with $Op(o)$, *a* uses *o* like the locking model *LM*, otherwise, *a* waits on *o*. □

[Example 4.3][Fig.3] In the state ($Op(o)$, *a*, $Wp(o)$) as presented in Example 4.2, suppose that *e* requests *o*, and $e \Rightarrow_M a$ and $e \Rightarrow_M b$, but neither $e \Rightarrow_M c$ nor $e \Rightarrow_M d$. $IO(Op(o), e) = \{c, d\}$ and $TO(Op(o), e) = \{c\}$. If $c \Rightarrow_M e$ and $d \Rightarrow_M e$, *e* is generally compatible with $Op(o)$ and then can use *o*. It is clear that a new state ($Op(o) \cup \{e\}$, *a*, $Wp(o)$) is safe. □

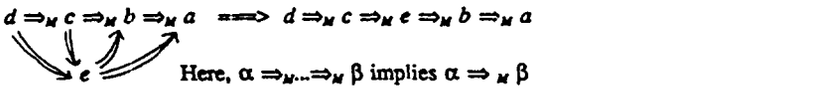


Fig.3 Extended locking Model *EM*

In Example 4.3, *e* is not compatible with $Op(o)$. Hence, *e* has to wait on *o* in the *LM*. Thus, the *EM* implies less number of waited operations. If the locking system is always safe, the compatibility among operations which are used for each object is consistent.

[Theorem 4.1] A lock system *L* is safe by the locking *LM* or *EM* and releasing *RM* schemes.
[Proof] It is clear that $Op(o)$ is empty or a singleton. Suppose that $LS(o) = (Op(o), hp(o), Wp(o))$ is safe. When an operation in $Op(o)$ releases *o*, it is clear the resultant locking system state is safe. When an operation *a* requests *o*, it is clear as shown in Example 4.2 and 4.3. □

Next, let us consider the compatibility relation on the compensate operations. First, the following axiom exists.

[Axiom] For every compensate operation a^- of an operation *a*, $a^- \Rightarrow_M a$. □

[Definition] a^- is said to be *subsumed* by *a iff* for every operation *b*, (1) if $a \Rightarrow_M b$, then $a^- \Rightarrow_M b$, and (2) if $b \Rightarrow_M a$, then $b \Rightarrow_M a^-$. □

[Example 4.4] In Fig.3, if $\{c, d\} \Rightarrow_M e \Rightarrow_M \{a, b\}$, $\{c, d\} \Rightarrow_M e^- \Rightarrow_M \{a, b\}$, and $e^- \Rightarrow_M e$, then *e* subsumes e^-. This means that e^- can use *o* when *e* uses *o* by the *EM*. □

[Theorem 4.2] If a^- is subsumed by *a*, a^- could use the object *o* when *a* uses *o* by the extended locking model *EM*.
[Proof] For every state $LS(o) = (Op(o), hp(o), Wp(o))$, if $a \in Op(o)$, a^- is generally compatible with $Op(o)$. in the *LM*, a^- can use *o*. □

4.2 Deadlock

We define what is the deadlock among nested transactions.

[Definition] For two different operations *a* and *b*, $a \Rightarrow_L^* b$ *iff* (1) for the *LM*, *a* in $Wp(o)$ and *b* in $Op(o)$, and (2) for the *EM*, (1) and *b* does not satisfy the general compatibility condition for *a*. □

A notation $a \Rightarrow_L b$ denotes that there exists some object *o* such that $a \Rightarrow_L^* b$. It means that *a* waits on *o* until every operation *b* such that $a \Rightarrow_L b$ releas s *o*. In this sense, our locking model is an *AND* model [KNAP87].

[Example 4.6] In Example 4.2, suppose that $e \Rightarrow_M \{a, b\}$ and not $e \Rightarrow_M \{c, d\}$. Suppose that $c \Rightarrow_M e$ but not $d \Rightarrow_M e$. *e* does not satisfy the general compatibility for *a*. Here, $e \Rightarrow_L d$. If *d* releases *o*, *e* can use *o* in the *EM*. But, in the *LM*, $e \Rightarrow_L \{c, d\}$. *e* cannot use *o* even if *d* releases *o*. *e* can use *o* only after both *d* and *c* release *o*. □

[Definition] $a \Rightarrow_L^* b$ (*a depends* on *b*) *iff* (1) $a \Rightarrow_L b$, or there is some operation *c* such that (2) $a \rightarrow_T^* c \Rightarrow_L b$, (3) $a \Rightarrow_L c \rightarrow_T^* b$, or (4) $a \Rightarrow_L^* c \Rightarrow_L^* b$. □

[Definition] *a* is said to be *directly deadlocked* in a system state *S iff* $a \Rightarrow_L^* a$. *a* is said to be *deadlocked* in *S iff* (1) *a* is directly deadlocked or (2) *a* depends on a deadlocked operation in *S*. *S* is said to be *deadlocked iff* there exists some deadlocked operation in *S*. □

[Example 4.7] Fig.4 shows a deadlocked state *S* for two transactions T_1 and T_2. In *S*, $e($ waits on an object *x* held by $f($, $j($ waits on *y* held by $c($. $e(\Rightarrow_L f($ and $j(\Rightarrow_L c($. $c(\Rightarrow_L^* e(\Rightarrow_L^* f(\Rightarrow_L^* j(\Rightarrow_L^* c($. Hence, $c($, $d($, *t*, $e($, $f($, *g*, $h($, *i*, and $j($ are directly deadlocked. □

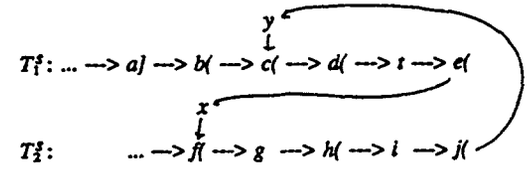


Fig.4 Deadlocked State

Here, an operation for which another deadlocked transaction waits is said to be a *waited* operation. In Fig.4, $c($ and $f($ are waited operations of T_1 and T_2, respectively. Let $W(T^s)$ denote a waited operation of *T*. In order to resolve the deadlocked state, one deadlocked transaction *T* is selected and operations from $C(T^s)$ to $W(T^s)$ are aborted. Although the whole transaction *T* is aborted in the conventional system, the deadlock can be resolved by aborting only part of *T*. This is a *partial abortion* of *T* [TAKI90]. A compensated subsequence $(T^s|_{W(T^s)})$ is said to be a *recovery sequence* $RS(T^s)$. For example, $RS(T_2^s) = \langle f(; g; h(; i; j(\rangle$. In this paper, $\langle f(; g; h(; i; j(\rangle$ is aborted by executing the compensate sequence, e.g. $\langle j); i^-; h); g^-; f) \rangle$.

5. SYNCHRONIZATION

We discuss relations among compensate sequences and lock/unlock operations.

5 ... Model

Let *Lock(a)* denote a set of objects held by an operation *a*. There are multiple kinds of computation models for executing *a*. Let *a* be an operation of an object *o* where $a^1 = \langle [a; a_1; ...; a_m; a] \rangle$. If *a* is executed in an *open- nested* scheme [GRAY81, TRAI83], the locks obtained by $a_1, ..., a_m$ are released by *a]* but the lock of *a* itself is not released.

[*Open-nested* execution of *a*] (1) Before executing *a* for *o*, *a* locks *o* by the *LM* or *EM*.
(2) *Lock(a)* := {*o*}. When $a_1, ..., a_m$ of *a* commit, all the locks in $Lock(a_1) \cup ... \cup Lock(a_m)$ are released by the *RM*, i.e. $Lock(a_j) = \phi$ for $j = 1, ..., m$. □

(2) means that *a* is two-phase locked [ESWA73], i.e. after some operation called by *a* releases a lock, any operation called by *a* requires no lock.

In the *close-nested* scheme [MOSS85, 86], all the locks obtained by a transaction *T* are released when the root of *T* commits.

[*Close-nested* execution of *a*] (1) is the same as the open-nested one.
(2) When $a_1, ..., a_m$ commit, $Lock(a) = \{o\} \cup Lock(a_1) \cup ... \cup Lock(a_m)$. If *a* is the root of the transaction, all the locks in *Lock(a)* are released by the *RM*. □

There is more general *semi-open-nested* scheme [TAKI90], where locks of some child operations are not released even if *a* commits.

[*Semi-open-nested* execution of *a*] (1) is the same as the open-nested one.
(2) When $a_1, ..., a_m$ commit, $Lock(a) = \{o\} \cup L_1 \cup ... \cup L_m$. Here, L_j is either $Lock(a_j)$ or ϕ for $j = 1, ..., m$. All the locks in $\cup_{j=1, ..., m} (Lock(a_j) - L_m)$ are released by the *RM*. □

Here, an operation a_j where $L_j = Lock(a_j)$ is said to be *closed* in *a*. If a_j is not closed, it is said to be *open* in *a*. If every child of *a* is closed and open, the execution of *a* is *close-nested* and *open-nested*, respectively.

5.2 Synchronized Sequence

Here, we consider relations among the nested transactions and locking mechanism. Let *a* be an operation on an object *o*. For a sequence *A*, a *synchronized* sequence (*S-sequence*) is one obtained by replacing *[a* and *a]* with synchronization operations, i.e. *lock* and *unlock* operations. It represents a sequence of operations and lock/unlock operations. Let *a(* denote a lock operation on *o*, i.e. *a(* means that $Lock(a) = \{o\}$. If *a* is a root or an action, *a(* denotes a *NULL* operation λ. Let *a)* denote an unlock on *o*, i.e. $Lock(o) = \phi$. If *a* is a root or an action, *a(* and *a)* denote a *NULL* operation λ. Suppose that $a^1 = \langle [a; a_1; ...; a_m; a] \rangle$. We introduce the following notations.

[Release operations] (1) $a)) = \langle a_1)); ...; a_m)); a) \rangle$. (2) $a]) = \langle a_1)); ...; a_m)) \rangle$. (3) $a]] = a])$ if *a* is a root, otherwise λ. (4) $a\rangle = a])$ if *a* is open, otherwise $\langle \Delta_1; ...; \Delta_m \rangle$ where $\Delta_j = a_j))$ if a_j is open, otherwise λ. □

a)) denotes unlocks of all the objects obtained in *a*. That is, every object in *Lock(a)* is

released. *a])* means that every object in $Lock(a_1) \cup ... \cup Lock(a_m)$ is released but the lock of *a* is not released. Thus, the open-nested execution of *T* requires less number of locks than the close-nested one. The open-nested execution can achieve more concurrency. Here, let *a[]* denote either *a]), a]],* or *a]>. a]), a]],* and *a]>* are said to be *open*, *close*, and *semi-open* release operations, respectively, which denote the last step (2) in the execution schemes as stated in 5.1.

[Definition] An *i-level open expansion* $A^{(i)}$, *i-level close expansion* $A^{[i]}$, and *i-level semi-open expansion* $A^{<>}$ of a sequence *A* are *S-sequences* obtained by (1) replacing *[a* with *a(*, and (2) *a]* with *a]), a]* with *a]]*, and *a]* with *a]>* in *A'*, respectively. □

The full expansions $A^{(*)}$, $A^{[*]}$, and $A^{<*>}$ are defined similarly to A^*. They are named *synchronized expansions* of *A*. Suppose that $a' = \langle [a; a_1^{[-1]}; ...; a_m^{[-1]}; a] \rangle$. $a^{(i)}$, $a^{[i]}$, and $a^{<i>}$ are given as follows.

[Synchronized expansion of *a*] (1) $a^{(i)} = \langle a(; a_1^{[-1]}; ...; a_m^{[-1]}; a] \rangle$.
(2) $a^{[i]} = \langle a(; a_1^{[-1]}; ...; a_m^{[-1]}; a]] \rangle$. (3) $a^{<i>} = \langle a(; a_1^{<-1>}; ...; a_m^{<-1>}; a] \rangle \rangle$. □

Here, $A^{((i))}$ is used to denote a synchronized expansion of *A*, i.e. $A^{((i))} \in \{A^{(i)}, A^{[i]}, A^{<i>}\}$.

[Example 5.1] In Fig.1, the open-nested execution of *T* is $T^{(*)} = \langle T(; a(; b(; c(; [[h]]; c]); [[d]]; b]); e(; [[f]]; e]); g(; [[i]]; [[j]]; g]); a]); T] \rangle = \langle a(; b(; c(; [[h]]; [[d]]; c); e(; [[f]]; g(; [[i]]; [[j]]; b); e); g); a) \rangle$. That is, locks of *a*, *b*, and *c* are obtained. Actions *h* and *d* called by *c* are executed. The lock of *c* is released since *b* commits. A lock of *e* is obtained, *f* is executed, a lock of *g* is obtained, then *i* and *j* are executed. Since *a* commits, the locks of *b*, *e*, and *g* are released, and then the lock of *a* is released. □

[Example 5.2] In Fig.1, the close-nested execution of *T* is $T^{[*]} = \langle T(; a(; b(; c(; [[h]]; c]]; [[d]]; b]]; e(; [[f]]; e]]; g(; [[i]]; [[j]]; g]]; a]]; T]] \rangle = \langle a(; b(; c(; [[h]]; [[d]]; e(; [[f]]; g(; [[i]]; [[j]]; a)); a) \rangle = \langle a(; b(; c(; [[h]]; [[d]]; e(; [[f]]; g(; [[i]]; [[j]]; c); b); g); e); a) \rangle$. Compared with Example 5.1, all the locks are released after *a* commits. □

[Example 5.3] In Fig.1, suppose that *c* and *b* are closed, and the other subtransactions are open. When *b* commits, the locks of *c* and *b* are obtained. When *a* commits, all the locks are released. That is, the semi-open nested execution $T^{<*>} = \langle T(; a(; b(; c(; [[h]]; c]\rangle; [[d]]; b]\rangle; e(; [[f]]; e]\rangle; g(; [[i]]; [[j]]; g]\rangle; a]\rangle; T]\rangle \rangle = \langle a(; b(; c(; [[h]]; [[d]]; e(; [[f]]; e]); g(; [[i]]; [[j]]; a]); a\rangle = \langle a(; b(; c(; [[h]]; [[d]]; e(; [[f]]; g(; [[i]]; [[j]]; c); b); e); g); a) \rangle$. Since *c* is closed, even if *b* commits, the lock of *c* is not released. □

5.3 Synchronized Compensate Sequence

Next, let us consider how to synchronize the compensate operations. For an *S-sequence A*, A^- is defined as follows. Here, suppose that $a^1 = \langle [a; a_1; ...; a_m; a] \rangle$ if *a* is a subtransaction.

[Definition] For a closed sequence *A*, (1) $A^{-1} = a^-$ if *A* is an operation *a*, and (2) $A^{-1} = \langle A_m^{-1}; ...; A_1^{-1}; a[] \rangle$ if $A = \langle a(; A_1; ...; A_m; a[] \rangle$. □

[Definition] For a closed sequence A, (1) $A^{-(} = a^-$ if A is an operation a, and (2) $A^{-(} = \langle a(; A_m^{-(}; ...; A_1^{-(}; a[\rangle$ if $A = \langle a(; A_1; ...; A_m; a[\rangle$ □

In $A^{-(}$, the lock operation of an operation a which is a parent of A is required since it is already released in A. On the other hand, A^{-1} does not require $a($ since A still holds it. It is noted that $a($ and $a^-($ are released in our model as soon as a is compensated by a^-.

[Definition] For an operation A, (1) $A^- = [[a]]^-$ if A is an action $[[a]]$, and
(2) $A^- = \langle a^-; a); a^- \rangle$ if a is a subtransaction. □

[Definition] For $A = \langle a(; A_1; ...; A_m; a[\rangle$ where $A_i >> a_i$ for $i = 1, \ldots, m$,
(1) $A^- = \langle A_m^{-(}; ...; A_1^{-(}; A^-]); a\rangle$ if $a[] = a]$) (open-nested),
(2) $A^- = \langle A_m^{-[}; ...; A_1^{-[}; A^-]]; a\rangle$ if $a[] = a]]$ (close-nested), and
(3) $A^- = \langle \Delta_m; ..., \Delta_1; A^- \rangle; a\rangle$ (where $\Delta_j = A_j^{-(}$ if A_j is open, $A_j^{-(}$ if A_j is close) if $a[] = a\rangle$ (semi-open-nested). □

Here, $A^-[]$ means the release of locks obtained in A^-, i.e. (1) $A^-[] = \lambda$ if A is an action, (2) $A^-[] = a))$ if $A = \langle a(; ...\rangle$, (3) $A^-[] = a^-))$ if A is an operation a.

[Example 5.4] (1) Let A be $\langle a(; b; c(; [[d]]; c]); [[e]]; a]\rangle$. $A^- = \langle [[e]]^-(; \langle c(; [[d]]; c]\rangle^{-(}; b^{-(}; A^-]); a\rangle = \langle [[e]]^-; c(; [[d]]^-; b^-; c); b^-); a\rangle$. After $\langle B; A; A^-\rangle$ is executed, not only $\langle B; A; A^-\rangle / \!\!\rightarrow_E B$, but also all the locks obtained in A and A^- are released.
(2) Let A be $\langle a(; b; c(; [[d]]; c]]; [[e]]; a]]\rangle$. $A^- = \langle [[e]]^-[; \langle c(; [[d]]; c]]\rangle^{-[}; b^{-[}; A^-]]; a\rangle = \langle [[e]]^-; [[d]]^-; b^-; c); b^-); a\rangle$. □

[Definition] For $A = \langle a(; A_1; ...; A_k; A_{k+1}\rangle$ where $A_i >> a_i$ for $i = 1, \ldots, k$, and $A_{k+1} < a_{k+1}$, $A^- = \langle A_{k+1}^-; A_k^{-(}; a_k)); A_k^-[]; ...; A_1^{-(}; a_1)); A_1^-[]; a\rangle$. □

[Example 5.5] (1) Let A be $\langle a(; b; c(; d; c]); e(; [[f]]\rangle$. $A^- = \langle\langle e(; [[f]]\rangle^-; \langle c(; d; c]\rangle^{-(}; \langle c(; d; c]\rangle^-]); b^{-(}; b^-]); a\rangle = \langle [[f]]^-; e); d^-; c); b^-; b^-); a\rangle$.
(2) Let A be $\langle a(; b; c(; d; c]]; e(; [[f]]\rangle$. $A^- = \langle\langle e(; [[f]]\rangle^-; \langle c(; d; c]]\rangle^{-[}; \langle c(; d; c]]\rangle^-]]; b^{-[}; b^-]]; a\rangle = \langle [[f]]^-; e)); d^-; c)); b^-; b^-)); a\rangle$. □

[Definition] For $A = \langle A_{j-1}; A_j; ...; A_m; a[\rangle$ where $A_i >> a_i$ for $i = j, \ldots, m$, and $A_{j-1} < a_{j-1}$,
(1) $A^- = \langle A_m^{-[}; A_m^-]]; ...; A_j^{-[}; A_j^-]]; A_{j-1}^-\rangle$ if $a[] = a]]$ (close-nested), and
(2) $A^- = \langle \Delta_m; ...; \Delta_j; A_{j-1}^-\rangle$ (where $\Delta_j = \langle A_j^{-[}; A_m^-]]\rangle$ if A_j is open, $\langle A_j^{-[}; a_m)\rangle$ if A_j is close) if $a[] = a]$ (semi-open-nested) and a_i is closed for $i = 1, \ldots, j\text{-}1$. □

In the open-nested execution, it cannot be defined, since a_i is already committed and locks held by the children of a_i are released.

[Example 5.6] Let A be $\langle b; c]]; d; [[e]]; f]]\rangle$. $A^- = \langle [[e]]^-[; [[e]]^-]]; d^{-[}; d^-]]; \langle b; c]]\rangle^{-[}\rangle = \langle [[e]]^-; d^-; d^-)); b^-; b)); b^-)\rangle$. □

[Definition] For $A = \langle A_{j-1}; A_j; ...; A_k; A_{k+1}\rangle$ where $A_i >> a_i$ for $i = j, \ldots, k$ ($k = j\text{-}1$ if a_j is open, $h = j$ if close), $A_{k+1} < a_{k+1}$, and $A_{j-1} < a_{j-1}$, $A^- = \langle A_{k+1}^-; A_k^-; ..., A_j^-; A_{j-1}^-\rangle$. □

[Example 5.7] (1) $\langle a; b(; c; d; b]); e\rangle^- = \langle e^-; e); e^-); \langle b(; c; d; b]]\rangle^-; a^-; a); a^-\rangle = \langle e^-; e); e^-); c^-; d^-; c^-); d^-); a^-; a); a^-\rangle$.
(2) $\langle a; b; c]]; e(; d; e]]; f\rangle^- = \langle f^-; f)); f^-)); \langle e(; d; e]]\rangle^-; \langle a; b; c]]\rangle^-\rangle = \langle f^-; f)); f^-)); d^-; d)); d^-)); b^-; b)); b^-)); a^-; a)); a^-)\rangle$. □

A synchronized compensate sequence A^- is obtained from an *S-sequence* A by using the following *CS* algorithm. Let R be a recovery sequence $RS(T^s)$, which contains m elements. Let R_i denote the i the element ($i = 1, \ldots, m$). Here, *push* and *pop* are operations on a stack, and *out* outputs compensate operations. The sequence output by the *CS* algorithm is a compensate $(RS(T^s))^-$.

[Compensate Sequence (*CS*) Algorithm] $st = \lambda$;
for $i = m, m\text{-}1, ..., 1$ /* backward reading */ do { $c = R_i$;
  if $c = [[a]]$, then *out*$([[a]]^-)$;
  else if $c = a$ /* subtransaction */, then { *out*(a^-);
    if $st = \lambda$, then if a is open, then *out*$(\langle a); a^-\rangle)$, else *out*$(\langle a)); a^-)\rangle)$; }
  else if $c = a[]$, then { if $st = \lambda$, then *out*$(a())$; *push*(st); $st = c$; }
  else /* $c = a($ */ if $st = \lambda$, then if a is open, then *out*$(\langle a]); a\rangle)$; else *out*$(a))$;
    else { *pop*(st); if $st = \lambda$, then if a is open, then *out*$(\langle a]); a\rangle)$; else *out*$(a))$;
      else *out*$(a[])$; } }
while $(b \neq \lambda)$ do *pop*(b); □

6. UNRESOLVABLE DEADLOCK

As presented before, the compensate operations are executed as transactions which require the locks on the objects. Problem is that another deadlock might occur while the compensate operations are executed in order to resolve the deadlock.

6.1 Problems

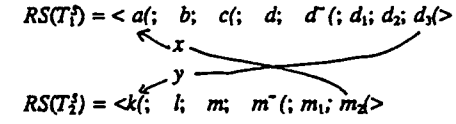


Fig.5 Unresolvable Deadlock

[Example 6.1] Suppose that $T_1^f = \langle a(; b; c(; d; e(\rangle$ and a deadlock cycle exists from $e($ to $c($ in a state S', and T_1 is selected to be aborted. $\langle c(; d; e(\rangle$ in T_1^f is partially aborted by executing $\langle e); d^-; c)\rangle$ where $(d^-)^{(1)} = \langle d^-(; d_1; d_2; d_3; ...\rangle$. $e)$ means that T_1 stops requesting the object. Then, the system is in a state S, $RS(T_1^f) = \langle a(; b; c(; d; d^-(; d_1; d_2; d_3(\rangle$ and $\langle k(; l; m; m^-(; m_1; m_2(\rangle$ where $(m^-)^1 = \langle m^-(; m_1; m_2; ...\rangle$. Suppose that m_1 requests the object x held

by $a($, and $d_3($ requests y held by $k($. In order to resolve the deadlock, one operation, say $a($, is selected to be aborted. First, $<d^{-}(; d_1; d_2; d_3(>^{-}$ is executed. Here, suppose that the abortion is successful. In order to abort d, d^{-} has to be executed again. However, the execution of d^{-}, i.e. $<d^{-}(; d_1; d_2; d_3; ...>$, implies the same deadlock as shown in Fig.5. Even if T_2 is partially aborted, the deadlock cannot be resolved in the same way. We call such deadlock *unresolvable* one [TAKI90]. □

We define what is the unresolvable deadlock. The following sets are defined for a state S.

$L(T^s) = \{ a \mid a(\in (T^s)^* \}$ = a set of operations which requested locks in T^s.
$U(T^s) = \{ a \mid a) \in (T^s)^* \}$ = a set of operations which released locks in T^s.
$UC(T^s) = L(T^s) - U(T^s)$ = a set of operations which lock objects in T^s.
$LI(T^s)$ = a compensate a^{-} whose lock $a^{-}($ appears lastly in $(T^s)^*$.
$N(T^s) = \{ a \mid a(\in (T^s|_l)^* \} - \{ a \mid a) \in (T^s|_l)^* \}$ (where $l = LI(T^s)$) = a set of operations which lock objects and are executed before $LI(T^s)$.
$I(T^s) = UC(T^s) - N(T^s)$ = a set of operations which lock objects and are executed after $LI(T^s)$.

[Definition] For two transactions T_1 and T_2, $T_1 \Rightarrow_U T_2$ (T_1 *unsafely depends* on T_2) in S *iff* (1) $I(T_1) \neq \phi$ and $I(T_2) \neq \phi$, and (2) for some operation a in $N(T_2^s)$, $C(T_1^s) \Rightarrow_L a$. Here, a is said to be an *unsafely waited* operation in T_2^s. $\Rightarrow_U^*$ is a transitive closure of $\Rightarrow_U$. □

[Definition] A transaction T is said to be *unresolvably deadlocked* in S *iff* $T \Rightarrow_U^* T$. S is said to be *unresolvably deadlocked iff* there is some unresolvably deadlocked transaction in S. □

[Example 6.2] In Example 6.1, a and k are unsafely waited. $LI(T_1^s) = d^{-}$, $UC(T_1^s) = \{a, b, c, d, d^{-}, d_1, d_2\}$, $N(T_1^s) = \{a, b, c, d\}$, and $I(T_1^s) = \{d^{-}, d_1, d_2\}$. $LI(T_2^s) = m^{-}$, $UC(T_2^s) = \{k, l, m, m^{-}, m_1\}$, $N(T_2^s) = \{k, l, m\}$, and $I(T_2^s) = \{m^{-}, m_1\}$. $I(T_1^s) \neq \phi$ and $I(T_2^s) \neq \phi$. $T_2^s|_k = <k(; l; m; m^{-}(; m_1; m_2(>$. $d_3(\Rightarrow_L k($. Therefore, $T_1 \Rightarrow_U^* T_2$. Also, $T_2 \Rightarrow_U^* T_1$. Since $T_1 \Rightarrow_U^* T_2$ and $T_2 \Rightarrow_U^* T_1$, T_1 and T_2 are unresolvably deadlocked. However, if $m_2(\Rightarrow_L d_1$, T_1 and T_2 are not unresolvably deadlocked. In this case, $<d^{-}(; d_1; d_2>^{-}$ can be executed and deadlock is resolved. □

[Theorem 6.1][TAKI90] If transactions are unresolvably deadlocked in the system M, it cannot be resolved by executing the compensate operations. □

[Example 6.3] (1) Let us consider two transactions T_1^s and T_2^s in a state S as shown in Fig.6. Here, $C(T_1^s) = j$ and $C(T_2^s) = y$. Suppose that every transaction is executed in an open-nested scheme. The locks of e, f, h, and v are released in S. Hence, if $j \Rightarrow_L t$, $T_1 \Rightarrow_U T_2$. Also, if $y \Rightarrow_L c$, $T_2 \Rightarrow_U T_1$.

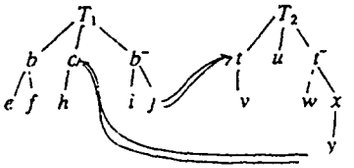


Fig.6 Unresolvable Deadlock

(2) Next, we consider a case that transactions are executed in a close- nested one. All the locks, e.g. e, f, h, and v are held. For example, if $j \Rightarrow_L v$, $T_1 \Rightarrow_U T_2$. If $y \Rightarrow_L h$, $T_2 \Rightarrow_U T_1$. Here, T_1 and T_2 are unresolvably deadlocked. □

As shown in this example, the close-nested execution has higher possibility that the unresolvable deadlock might occur than the open-nested execution. It depends on the execution scheme which operations might be unsafely waited operations.

6.2 Safe System

[Theorem 6.2] If every transaction T is executed in a close-nested scheme and the compensate of the fully expanded recovery sequence of T, i.e. $((RS(T^s))^{[*]})^{-}$ is executed, no unresolvable deadlock could occur.
[Proof] Let S be a state of the system M. Since T is executed in a close- nested scheme, $(RS(T^s))^{[*]}$ includes no unlock operations. Hence, $((RS(T^s))^{[*]})^{-}$ includes only the compensate of actions and unlocks. Since no locks are required in $((T^s)^{[*]})^{-}$, no unresolvable deadlock would occur. □

Here, it is noted that the unresolvable deadlock might occur if $(RS(T^s))^{[*]}$ is undone by executing another higher sequence than $((RS(T^s))^{[*]})^{-}$. Because a compensate A^{-} of a sequence A such that $A >> (RS(T^s))^{[*]}$ may require locks.

[Theorem 6.3] If (1) the system M is minimum acyclic, (2) every operation subsumes its compensate and M adopts the extended model EM for the compensate operations, and (3) every transaction is executed in an open-nested scheme, no unresolvable deadlock would occur.
[Proof] From theorem 4.2, every compensate operation a^{-} can use the object when a uses it. Since every transaction T_1 is executed in an open-nested scheme and M is minimum acyclic, there is no operation b in $N(T_1^s)$ such that $C(T_1^s) \Rightarrow_S^* b$ in every state S. Hence, there is no transaction T_2^s such that $T_1 \Rightarrow_U^* T_2$ and $T_2 \Rightarrow_U^* T_1$ in S. □

However, if some transaction is executed in a close-nested scheme, some unresolvable deadlock might occur. Because $N(T^s)$ can include some operation b such that $C(T_1^s) \Rightarrow_S^* b$ since the locks are not released until the root commits. Also, if some compensate operation a^{-} is not subsumed by a, unresolvable deadlock might occur as shown in Example 6.4. A system M which satisfies the conditions (1), (2), and (3) in Theorem 6.2 is said to be *safe*.

[Example 6.4] Suppose that some transaction T_1 calls file operations f_{11}, f_{12}, and f_{13}. Suppose that when f_{13} is called, some deadlock occurred and f_{12} is undone by executing f_{12}^{-}. Also, a transaction T_2 calls f_{21} and f_{22}, and f_{21}^{-} is executed to undo f_{12}. Here, suppose that f_{12}^{-} is not compatible with f_{21} and f_{21}^{-} is not compatible with f_{11}. If every compensate operation is generally compatible and the extended locking model LM is used, neither $f_{12}^{-} \Rightarrow_L^* f_{21}$ nor $f_{21}^{-} \Rightarrow_L^* f_{12}$, because if f_{12} uses the object, f_{12}^{-} must use it. However, if f_{12}^{-} is not subsumed by f_{12} and is not compatible with f_{21}, $f_{12}^{-} \Rightarrow_L^* f_{21}$. Also, $f_{21}^{-} \Rightarrow_L^* f_{121}$. Unresolvable deadlock occurs. □

6.3 Unsafe System

Let us consider a more general system which is not safe. [TAKI90] shows one prevention method named *semi-open* one that an operation a is executed in a close-nested scheme if a^- has some possibility to cause the unresolvable deadlock, although most operations are executed in an open-nested scheme. This is based on an assumption that every operation calls only operations in a predefined set of operations. When M is compiled, the relations $\rightarrow_c$ and $\Leftrightarrow_s$ are fixed and then so is $\rightarrow_s$. However, if each operation a calls operations dynamically, the predefined set of a cannot be defined before a is executed. Theorem 6.3 says that unless the conditions (1), (2), and (3) hold, some unresolvable deadlock might occur. In this paper, one detection method is presented for general systems which provide some special *super-compensate* operations.

[Definition] An operation b is said to be a *super-compensate* operation of a *iff* for every state s and every intermediate state t obtained by applying a to s, $b(t) =_g s$. □

Let a^{--} denote some super-compensate operation of a. That is, even if a stops before committing, we can get an equivalent state by applying the super-compensate a^{--}. One method to realize a^{--} is a conventional checkpointing. If some failure occurs, we can go back to the most recent checkpoint. a is said to be *super-compensatable* in T *iff* a has its super- compensate or has a super-compensatable ancestor. T is said to be *super- compensatable iff* every operation is super-compensatable in T. Our method is a detection method. If some unresolvable deadlock is detected, one unresolvably deadlocked transaction T is selected. Here, let a be the unsafely waited operation of T and b be $C(T^s)$.

[Abortion of $(T^s|^a_s)$] Let c be a super-compensatable operation which is an ancestor of b.
(1) If $c \rightarrow_c^* a$, then c^{--} is executed and then T is re-executed from c.
(2) Otherwise /* c is not an ancestor of a */, $((T^s|^a_s)^{-s} - \{c\})^-$ is executed after c^{--} is executed, and T is re-executed from a. □

If every operation in T is super-compensatable, any unresolvable deadlock can be resolved. However, unless some transaction is super-compensatable, unresolvable deadlock cannot be resolved.

7. SELECTION OF COMPENSATE OPERATION

If an operation a is compensatable, a may have multiple compensate operations, i.e. $|C_g(a)| \geq 1$. In order to undo a, one compensate operation a^- has to be selected. Another problem is that for a sequence A, there may be multiple compensate sequences. That is, for every sequence B equivalent to A, i.e. $B \in [A]$, B^- is a compensate of A. First, suppose that for every operation a including lock/unlock, $cost(a)$ gives a cost for executing a. For a sequence $A = \langle a; B\rangle$, $cost(A) = cost(a) + cost(B)$. For an operation a, let a^{-min} denote a compensate operation b of a such that $cost(b)$ is minimum in $C_g(a)$.

[Definition] A *minimum compensatable* sequence a^{min} of a is defined as follows.
(1) $a^{min} = a$ if a is an action or lock/unlock, (2) $a^{min} = a$ if $cost(a^{-min}) < cost((a^1)^{-min})$, and (3) $a^{min} = \langle a[; a_1^{min}; ...; a_m^{min}; a[]\rangle$ otherwise. □

For a sequence $A = \langle a; B\rangle$, A^{min} is defined to be $\langle a^{min}; B^{min}\rangle$. $((RS(T^s))^{min})^-$ is executed to abort $RS(T^s)$ with the minimum cost. In real systems, it is a problem how to compute $cost(a)$ for each operation a. $cost(a)$ is computed as the estimated value based on the statistic information.

8. CONCLUDING REMARKS

In this paper, we have discussed what is the compensate operation and how to use them. We have discussed the relations among the lock synchronization mechanism and the compensate operations. We show that there exists some unresolvable deadlock which cannot be resolved by executing the compensate operations. Also, under some constraint, i.e. *safe* condition, any unresolvable deadlock can be prevented. However, in general systems, unresolvable deadlock might occur. We have shown one solution for a general system by introducing the *super-compensate* operations. This method is a detection one. After the unresolvable deadlock is detected, one transaction is selected and partially aborted by using the super-compensate operations. This paper gives the foundation for aborting transactions by using the compensate operations.

REFERENCES

[BEER89] Beeri, C., Bernstein, P. A., and Goodman, N., "A Model for Concurrency in Nested Transaction Systems," JACM, Vol.36, No.2, 1989, pp.230- 269.

[BERN87] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," Addison Wesley, 1987.

[CHAN83] Chandy, K. M., Misra, J., and Haas, L. M., "Distributed Deadlock Detection," ACM TODS, Vol.1, No.2, 1983, pp.144-156.

[CHAN85] Chandy, K. M. and Lamport, L., "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Trans. on Programming Language and Systems, Vol.3, No.1, 1985, pp.63-75.

[ESWA76] Eswaren, K. P., Gray, J., Lorie, R. A., and Traiger, I. L., "The Notion of Consistency and Predicate Locks in Database Systems," CACM, Vol.19, No.11, 1976, pp.624-637.

[GARZ88] Garza, J. F. and Kim, W., "Transaction Management in an Object- Oriented Database System," Proc. of the ACM SIGMOD Conf., 1988, pp.37- 45.

[GRAY81] Gray, J., "The Transaction Concept: Virtues and Limitations," Proc. of the 7th VLDB, 1981.

[HAER83] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Database Recovery," ACM Computing Surveys, Vol.5, No.4, 1983.

[KIM89] Kim, W., Bertino, E., Garza, J. F., "Composite Objects Revised," Proc. of the ACM SIGMOD Conf., 1989, pp.337-347.

[KNAP87] Knapp, E., "Deadlock Detection in Distributed Databases," ACM Computing Sur-

veys, Vol.19, No.4, 1987, pp.303-328.
[KORT83] Korth, H. F., "Locking Primitives in a Database System," JACM, Vol.30, No.1, 1983, pp.55-79.
[LYNC86] Lynch, N. a· . Merritt, M., "Introduction to the Theory of Nested Transactions," MIT/LCS/TR 367, 1986.
[MOSS85] Moss, J. E., "Nested Transactions: An Approach to Reliable Distributed Computing," The MIT Press Series in Information Systems, 1985.
[MOSS86] Moss, J, E., Griffeth, N. D., and Graham, M. H., "Abstraction in Concurrency Control and Recovery Management(revised)," TR COINS 86-20, Univ. of Massachusetts, 1986.
[SING89] Singhal, M., "Deadlock Detection in Distributed Database Systems," IEEE Computer, No.11, 1989, pp.37-48.
[TAKI90] Takizawa, M. and Hyoudou, A., "Unresolvable Deadlock of Nested Transactions by Partial Abortions," submitted to publication, 1990.
[TRAI83] Traiger, I. L., "Trends in System Aspects of Database Management," Proc. of the 2nd International Conf. on Database (ICOD-2), 1983, pp.1- 21.
[WEIH89] Weihl, W. E., "Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types," ACM Trans. on Programming Language and Systems, Vol.11, No.2, 1989, pp.249-283.
[WEIK86] Weikum, G., "Theoretical Foundation of Multi-level Concurrency Control," Proc. of the 5th PODS, 1986, pp.31-42.

COOPERATING AGENTS - A DATABASE PERSPECTIVE

S.M. Deen
DAKE Centre
University of Keele
England

Email : deen@uk.ac.kl.cs

Abstract

A Cooperating Knowledge Based System (CKBS) is a collection of autonomous and potentially heterogeneous, objects (or units) which cooperate together in solving problems. An object can be a data/rule base or an agent which can be intelligent or unintelligent. The paper investigates the issues in CKBS from a database perspective, which is distinguished from the AI perspective by its emphasis on a common architecture with well-defined components and interfaces, performance efficiency and concurrency/reliability/recovery. The paper describes a generic object model, and also a hybrid cooperation model which incorporates the blackboard approach within the framework of a distributed transaction. The object model has a layered structure, complete with the necessary software components for run-time execution. The hybrid cooperation model is claimed to reduce unnecessary communications, while retaining the effectiveness of a blackboard approach. Three exploratory applications - travel planning, image recognition and air-traffic control - have been used to illustrate this approach.

The concepts presented in this paper are being further developed in the COSMOS project, in which a number of British Universities and Polytechnics, collaborate.

1 INTRODUCTION

Recent research activities in AI and databases seem to have established two important facts: (i) there are some common problem domains where a dialogue between AI and DB communities is fruitful, and (ii) there are many industrial applications of AI where some database facilities are required. Several workshops organised by M. Brodie et al [1] and the numerous research projects on data/knowledge base integration [2] bear testimony to these twin assertions. It would appear to us that the topic of Cooperating Knowledge Based Systems (CKBSs) as defined below, constitutes a further common domain where both the communities, particularly the researchers in distributed AI (DAI) and distributed databases (DDBs) can fruitfully cooperate.

Two autonomous systems (as nodes) may be said to cooperate in problem solving if they have to exchange intermediate results in solving the problem. Since many such systems require volume processing of data and rules, they are of interest to database researchers. From the database perspective, we are interested in the following three issues:

1. General Architecture

   We would like to develop a general architecture with well-defined interfaces, abstraction levels and components, so that such a system can be used for a class of applications, rather than a single application. A formal framework will allow the development of standardised facilities.

2. System Performance

   Within this general framework we would like to design a system that is operationally efficient. A cooperating system is expensive in communications terms. The need to exchange partial results can be reduced by providing each node with some basic information about the other, and by pre-planning the solution steps. This is a strategy we generally follow in distributed databases; however it may not be necessarily consistent with a pure AI approach. There is however a class of problems, such as in image recognition from raw data (signals from sensors), where complete global information cannot be realistically supplied to each node. Sensor data have to be divided into subareas and possibly abstracted for data reduction, and this leads to what we would refer to as *boundary problems*. Heuristics or some other algorithms will be required to resolve these conflicts, possibly using data from overlapping subareas. While the design of the these algorithms may not be considered as database research, we must provide anchor points in our architecture where these algorithms can be hooked.

   Another possibility that one might explore is to design nodes in such a way that refinements of intermediate results are carried out by higher-level nodes, thus reducing internodal communications. In that event the problem will be similar to that of distributed databases, but it is not a practicable proposition in many applications.

3. Concurrency, reliability and recovery

   These are essential facilities that an operational system must have, and for which a proper architectural framework is needed, as proposed in this paper. Apart from this architectural framework, we shall not consider these issues explicitly.

Having indicated our database perspective, we would define a Cooperating Knowledge Based System (CKBS) as a collection of autonomous, potentially heterogeneous and possibly pre-existing, objects (or units) which cooperate in problem solving in a decentralised environment. An object can be a knowledge server (a data/rule base, sometimes referred to here as knowledge base) or an agent as in a multi-agent system (MAS) [3]. An agent can be intelligent or dumb. While an agent may have its own private belief, a knowledge server holds

shared knowledge. In our approach a CKBS, like a database, is more concerned with practical usability and performance as outlined earlier, rather than the purity of ideas and their faithful representation; this apart a CKBS is similar to a MAS. Our reason for the definition of CKBS is not to re-invent the wheel, but to establish a context for this paper, independent of the DAI concepts of multi-agent system. We believe that some of the concepts developed in distributed databases could be useful to MAS in general, and CKBS in particular. We shall not make any distinction between the terms Cooperative Distributed Problem Solving (CDPS) systems of Durfee et al [4] and MAS.

We have recently undertaken a project called Coordination Strategies for Multi-object Systems (COSMOS) to explore this area. *Cooperation* may be viewed as a special case of *coordination* where the objects interact in a friendly rather than in a hostile environment. Our approach in COSMOS is based on a generic object (agent) model and a hybrid cooperation framework which incorporates blackboard ideas within a concept of distributed transaction[20], as discussed in this paper.

However we shall begin with a review of current research and our own ideas in section 2, followed by our object (agent) model in section 3 and hybrid cooperation framework in section 4. The effectiveness of our approach will be illustrated in section 5, with a number of diverse but exploratory applications: one main application from travel services and two supporting applications from air-traffic control and image recognition. A conclusion is given in section 6.

2 REVIEW

Research in distributed Artificial Intelligence (DAI) began in earnest from 1980 onwards, although there were also some earlier projects, such as the HEARSAY speech system [5]. Recently a number of prototypes have been proposed in multi-agent systems: among these are the DVMT of Durfee et al [6,7], Contract-Net of Davies and Smith [8,9], Actors model of Agha et al [10,11], Air-traffic control models of Steeb et al [12-14], MACE system of Gasser et al [15], and Agora of Bisiani et al [16]. Some of them have been implemented as prototypes and many of them are based on the blackboard architecture [17] for cooperation.

Despite this large number of models there has not been much attempt to unify the field of MAS with an integrated perspective [3], except some generalised blackboard models [27-29]. Agent modelling is a case in point. Although many of these projects use some form of agent descriptions, they do not propose any comprehensive model of an agent; instead they often represent only the actions of an agent. Kamel and Syed [18], discuss an interesting object-oriented approach for MAS, but do not suggest any agent models; likewise Bond [19] indicates some behavioral characteristics of agents, but does not go any further.

Major achievements in heterogeneous distributed databases have been the development of homogenisation and integration facilities [21,25,26] backed by a sound transaction model[20], to an extent that commercial products can now be built, and are indeed appearing on the market [23]. Although a node in a distributed database is assumed to be autonomous, and sometimes heterogeneous, there is no active cooperation except subquery executions and onward transmission of results. The characteristics of DDB are:

1. A DDB handles only knowledge servers but not agents (as defined earlier).
2. A DDB is a general purpose facility to retrieve and update data in the knowledge servers; it does not automatically produce global solutions purely from local computation without a global schema (or equivalent tables). An exception to this is a multi-database system where there is no global schema, but the user is assumed to provide the link [22, 25].
3. There is a unique answer to a query.

In contrast, we may define the characteristics of our CKBS, borrowing some ideas from [13], as:

1. It handles both agents and knowledge servers, which are generally heterogeneous.
2. Objects and skills may display many to many relationships, with some objects having overlapping, even identical, capabilities.
3. Each object has an immediate circle of acquaintances which may be different from those of another object of even the same type. This set of acquaintances may be different from those of another object.
4. An agent may have both private and shared knowledge.
5. An object decomposes a problem on the basis of some knowledge of the environment and it allocates the tasks on the basis of its knowledge of its immediate acquaintances. However, this knowledge of the environment and the acquaintances may not be complete or up-to-date.
6. If a receiving agent is unable to solve an allocated problem, it may pass it to an acquaintance who can.
7. A receiving agent may reason about the information it has received and enter into a dialogue with the sender or other relevant agents.
8. The agents converge on a solution in accordance with a specified criteria.
9. There may not be any unique solution to a problem.

Three interesting criteria of a multi-agent system (apart from decentralised control and potential heterogeneity) seem to be:

1. Each suitable agent should be able to solve its problem in consultation with other agents as needed.
2. There is no centralised global view point[3].
3. Globally coherent solutions must be obtained through local computation alone [3].

These characteristics have a number of consequences. The limitation of partial global knowledge is generally offset by increased inter-agent communications. The less the global knowledge, the more is the need for communications, but the communication cost can be unacceptably high[13].

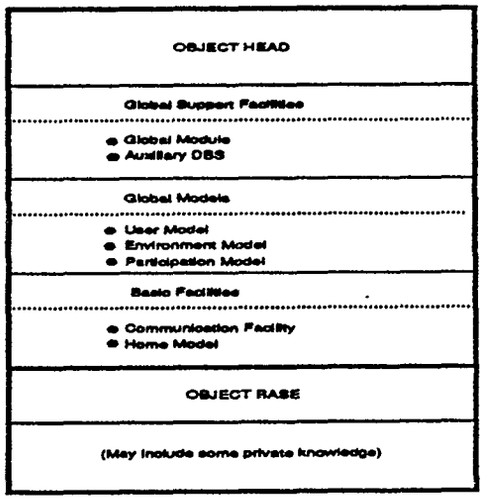
We, however, accept these criteria except that we would like to feel free to include a comprehensive global view point (as in distributed databases) as part of each object model, if this improves the overall system performance (taking the resultant update overheads into account). To reduce inter-object communications, particularly when there are too many objects communicating with each other, we use superior objects (in an aggregation hierarchy) to control this communication cost where appropriate. This hierarchy is problem-dependent, and can be defined automatically during the task decomposition/allocation stage of each new problem as needed. Reasoning can be carried out in two stages: at the task allocation stage (pre-event), and at the acceptance stage, i.e. when the result is presented (post-event). Whenever possible, we opt for pre-event reasoning (with the help of global information), as it reduces communication cost.

We support global information in a decentralised form; each object can learn and retain its new knowledge, although the global knowledge may not be identical in each case. There is an associated update problem which we view as minor compared to the improvement in performance this offers. We also recognise that all the necessary information (as global knowledge) on other objects (foreign objects) may not be always available, and this might require a search to locate a suitable foreign object, with additional communication overheads.

In our case all computations are object-based (i.e. local) and global coherence is partly enforced by pre-event reasoning during the task decomposition process, subject to some final confirmation by post event reasoning, where relevant. In a hierarchic structure, each higher-level object is responsible for the subsequent decomposition of its task into subtasks for its lower-level objects. In this scheme the resolution of *boundary problems* , as indicated earlier, is left to the individual applications, where appropriate objects will invoke suitable algorithms as needed. The task execution plan (see later) will ensure that an object gets the partial results it needs from other relevant objects, at a minimal communication cost.

3 OBJECT MODEL

We propose a generic framework for an object model (figure 1), based on our earlier work in distributed databases [22].
In this framework each object contains a base which is the basic autonomous system and a head which provides the additional facility needed for cooperation . The head has three main components as listed below:



Object Module (Figure -1)

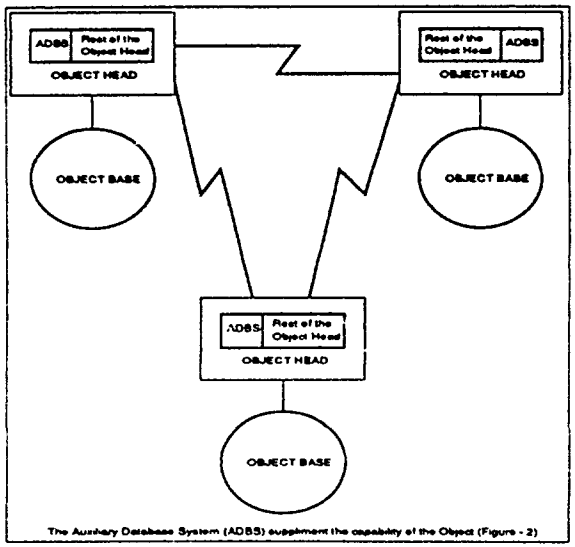
1. Global Support Facilities
   - Global Module (run-time software)
   - Auxiliary Database System (metadata and supplementary functions)
2. Global Models
   - User Model (facilities visible to end user)
   - Environment Model (the model of the operational environment)
   - Participation Model (models of the participating objects)
3. Basic Facilities
   - Communication facility (to communicate with other objects)
   - Home Model (the model of this object)

Each object in our scheme is autonomous and reusable, with an open interface, which enables it to participate in more than one cooperating system with appropriate upper levels. An object must also support inter-operability. Each object (the home object), has an immediate circle of acquaintances (foreign objects); the home and the foreign objects together participate, as participating objects, in cooperating problem solving. Although an object can be the

duplicate of another object, it may not necessarily have the same set of acquaintances. Each acquaintance, in its turn as a home object, may have a different set of acquaintances. There could be many levels of cooperation in which an object can participate. In addition, an object can be hierarchic (as in aggregation): an object (superobject) being made up of other objects (subobjects). For instance in air-traffic control, regional controllers cooperate together, but a regional controller itself could represent a lower-level cooperative system among flight-path allocators, transit controllers, airport-arrival/departure controllers and monitoring stations. A monitoring station could be a hierarchic object consisting of a collection of radars and other monitoring equipment.

Global Support Facilities

The Global Module consists of the software needed for the global operation, whereas the private auxiliary database system (ADBS) (figure 2) holds the directory information and supplementary operational facility. This facility is essential if we are to provide open interfaces to support pre-existing objects. For instance if a travel agent (software) X can yield all the flights between points A and B in a single query, and another travel agent Y can yield only one flight per query, then the ADBS of Y could simulate the capability of X by generating one query per flight from the original single query.



The global module is equivalent to the run-time control system of the DDB, and will include all the necessary software to make the cooperating system work. The facilities supported by the auxiliary database system include:

- global models descriptions
- cached data/knowledge from knowledge servers
- replicated data/knowledge for knowledge servers
- integration information and integration facilities[21]
- special operations to cater for operations that are not available at this object

In a distributed database this role is fulfilled by a supplementary or subsidiary database[22]. Integration information will provide missing information backed by integration facilities as elaborated in the next section.

Global Models

The *user model* specifies the facilities of the cooperating system visible to an end-user; the user-task is specified in terms of these facilities. A user model has two parts: a task specification model, which is used to specify a task, and a response presentation model, which is used to present responses to the end user. All user queries are validated against this model.

The *environment model* provides a description of the operational environment which consists of object classes, their inter-relationships and capabilities, including functions and procedures that can be invoked for integration. The contents of both the environment and participation models are likely to be partial, and may not always be up-to-date; they can also vary dynamically (see later).

The *participation model* (*cf.* participation schema of distributed databases) consists of entries for all participating objects, that is the home object and its immediate circle of acquaintances. It is an instantiation of the environment model, with items taken from the home models of the participating objects. It may also have special fields which hold experiences of the home object in dealing with its acquaintances. All items of a home model may not be relevant to a particular cooperation strategy, and therefore we insist in having a participation model of the home object. This permits the home model of a home object to be defined independently to the global models.

The information of participation models can be held in two different ways[24]:

- Cache
- Replicated data

If the information is held as cache, it cannot be updated automatically, except through a cache update policy[24]. On the other hand if it is held as replicated data, then one can use a weak consistency in update[22].

Basic Facilities

The communication facility consists of a set of protocols, appropriate to the object and coordination model. These protocols should be designed as standard, to be transmitted over a file server within the ISO/OSI framework.

The home model is an abstract description of an object and its characteristics, and all access to the object is directed only through this home model, which acts as the encapsulation barrier. The home model is equivalent to local (nodal) schema of a distributed database and it contains data types for the properties and information content of the object concerned. Normally only a subset of these properties appear in the participation model.

The home model of a hardware based agent will contain all its relevant physical properties, including information on its skills, performance, techniques it uses, reliability, faults, faults diagnostics, potential application environment, and relationships to other agents (e.g. duplicate capability).

An object base along with its home model and the communication facility is known as a basic object, and is independent of any specific coordination environment. A basic object can participate in any suitable cooperating system, with appropriate upper levels (global models and global support facility). A basic object may also represent, hidden from the upper layers, a hierarchy or another level of cooperation, in which case its home model will act as the global model of the lower-level objects. A lower-level object can sometimes be a pure basic object, without the upper layers, e.g. a sensor.

An agent may or may not be intelligent, depending on the content of its head, which may vary from agent to agent. Typically a sensor as an agent is unlikely to be intelligent.

4 COOPERATION FRAMEWORK

We would like to group the cooperation strategies for the application domains of our interest into the following classes:

- Transaction oriented strategy
- Recognition oriented strategy
- Control oriented strategy

Although carried out independently, our classification is very similar to the one given in ref [4]. In a transaction oriented strategy, the objective is to produce new information, such as data, rules, text or image, using a top-down approach , in which each task is decomposed into subtasks recursively - finally into small granules that can be executed by an object. Any appropriate object can initiate the processing. Some examples of a transaction oriented strategy are: a complex query, e.g. for a travel plan where several expert systems have to cooperate to produce the answer; response integration where several expert systems discuss an issue to reach a consensus; and iterative design where several expert systems cooperate in producing a final design by iterative refinement from a rough plan.

In a recognition oriented strategy, the objective is to identify/recognise patterns from some input which could be raw data or monitoring data. This strategy uses a bottom-up approach, in contrast to the transaction oriented strategy; it also causes the boundary conflicts mentioned earlier. Examples of this strategy are the DVMT project of Durfee et al [6] and the Multi-agent vision system of Lane et al [30] presented in this conference.

A control oriented strategy differs from a transaction strategy in that its objective is to control an operation so that it can be completed successfully, in spite of potential distractions, possibly in a hostile environment. It often involves planning and sometimes forecast and negotiation. Typical examples are: air-traffic control, network traffic management, automated process control (with robots) and guided missiles.

Although these strategies are diverse requiring different types of solution, we claim that the agent model proposed earlier can be used as a general facility in all of these. Following this idea of generalisation we present below a hybrid cooperation framework which also can be used by all three strategies. We shall first develop this framework in terms of the transaction-oriented strategy alone, and then in the next section we shall show how it can be adapted for the other two strategies as well.

The blackboard framework represents a general technique for task decomposition, scheduling and inter-agent information exchange, but it is communication-intensive[3], even in a multi-panel scenario, since the agents have to communicate through their respective panels. In contrast in a transaction model of a DDB the decomposed tasks are allocated to appropriate nodes depending on their abilities. These nodes can use other nodes if needed, in a number of ways [22]. This is achieved in a DDB by holding the complete global knowledge in the form of a global schema (or equivalent), but in a CKBS we do not have a complete global view.

We propose here a hybrid scheme, incorporating ideas from both the blackboard framework and transaction models with the following main changes to the blackboard approach:

1. A tentative schedule (an execution plan) is prepared in an optimal manner by using the partial global information available. A dynamic strategy stipulates subsequent actions, where to find/send intermediate results, and what to do if the allocation is incorrect.
2. The relevant acquaintances communicate with one another directly (bypassing their panel), guided by the specification in the schedule.
3. A low-cost facility exists for each object to learn more about the environment and acquaintances, and thus to update the global knowledge. This allows the object to offset the effect of partial and out of date global information.

Our main objective above is to reduce the communication cost of the blackboard framework, while retaining its flexibility. We view the communication network as a black box, as assumed in distributed databases. Since each object is encapsulated with possibly private knowledge, we do not need to provide any facility for the preservation of global consistency in updates for these objects. A diagram for our hybrid model is given in figure 3, which is explained below briefly.
The model has the following components:

1. The **Task Manager** which manages a task (i.e. a user request or query) and is in overall control. It may support different coordination strategy for specific situations.

2. The **Higher Decomposer** which decomposes a task into subtasks using the environmental model. Here only object types are used.

3. The **Higher Recomposer** which produces a presentation format for the final answer.

4. The **Lower Decomposer** which decomposes the subtasks into nodal tasks, using the participation models. Here object types are instantiated by actual objects.

5. The **Lower Recomposer** produces a recomposition and response plans, using integration tools. It describes how the final result is produced.

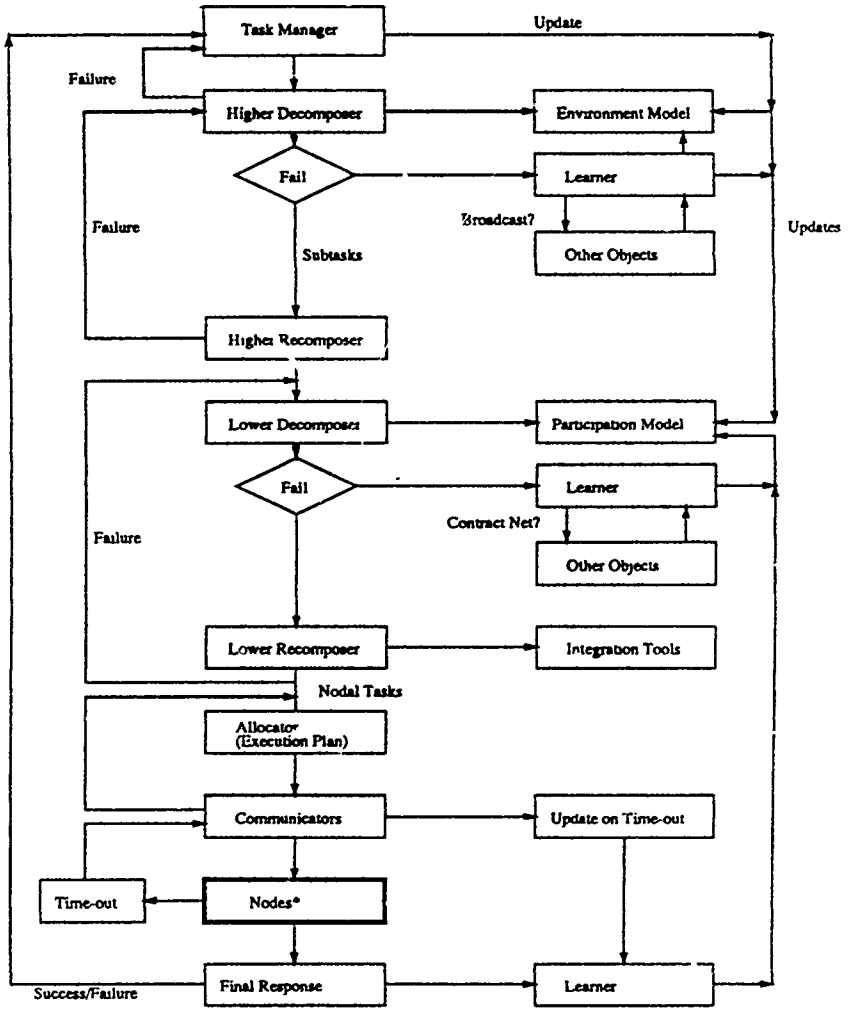
6. The **Allocator** produces an execution plan (made up of many subplans) and allocates nodal tasks optimally, taking network performance into account.

7. The **Communicator** supports protocols and provides the inter-object communication facility with a time-out condition. A message may include a dynamic coordination plan if supported by the Task Manager.

8. There are a number of **Learners** at various levels which are invoked if a decomposition is unsuccessful and also if a task (or its decomposed components) succeeds/fails (to acquire experiences).

9. Integration facilities consists of an **Integrator** for data/knowledge integration and conflict resolution. An integrator produces a final form of the integrated results using converters and mediators. A converter is a simple function which carries out reversible conversions, such as mile to kilometer, while a mediator is a complex procedure which might lead to a compromise solution. It may not have a inverse.

Learning at the decomposition stage can be accomplished either by a cache or replication policy. In this paper we assume learning by cache which requires re-cacheing. In this case, data can become out of date, and a periodic re-cacheing policy is needed. However, if replication is used instead, the owner nodes (from which data are copied) will automatically send out update messages whenever any change has occurred. Returning to the diagram, the user task is formulated using the user model. The task is received by the Task Manager which produces a higher-level decomposition, followed by a high-level recomposition strategy. If the Task Manager cannot decompose the task successfully, it invokes a Learner to update the environment model before a retry. Such a situation would arise only if the facilities indicated in the user model are not supported by the environment model. When a Learner updates the environment model, it also updates the participation model if appropriate. The update of a participation model could involve the addition of new objects, deletion of old objects, or amendment to existing objects.



Hybrid Cooperation Strategy (Figure -3)

The resultant subtasks are decomposed into nodal tasks. If this decomposition fails, then the Learner is used to update the participation model (i.e. to find new acquaintances), as needed before a retry. The Learner could use a Contract-Net protocol, depending on its strategy. Execution plans and response plans are also generated by the Allocator and Recomposer. The Communicator distributes the nodal tasks, with a time-out. If the time-out expires, the Communicator returns to the Allocators for fresh allocations.

It is assumed that if well-defined separate components as indicated above are used, then these aspects of the model can be changed easily.

Cycles, Concurrent usage, Updates and Extensibility

If a selected acquaintance (first level) cannot perform an allocated task, it can seek assistance from its own acquaintances (second level). This process can continue until an acquaintance that can do the task is found, subject to a time-out constraint. If the search is successful, the participation model of the original requester object will be updated, adding this new acquaintance. However, such a search can lead to cycles, which can be avoided by using hierarchical identifiers for each nodal task[24]. Our model will support concurrent usage facility (multi-user environment), but further work is needed to design appropriate algorithms.

As stated earlier, when a Learner updates an environment model, it also updates the relevant participation models, with new acquaintances. If a task succeeds or fails, further updates can be made depending on information held on the models. This allows experiences to be gathered. If the task fails the control returns to the Task Manager, which may update some statistics at various levels. If the time-out expires, it leads to a fresh allocation of tasks, but if the allocation fails, then the Lower Decomposer is invoked, which may in turn invoke a Learner. When the Higher Recomposer fails, the Higher Decomposer is invoked, which in turn might invoke the Higher Learner.

It may be observed that the communication cost for Learning is low, since the Learners are invoked only rarely at task decomposition. The total communication cost could be up to 50 percent lower than in the corresponding framework as shown in figure 5 of section 5.1.

The user should be able to define new data types and functions (abstract data types) with the help of those available in the user, environment and participation models. It should also be possible to create new agents dynamically.

5 APPLICATION EXAMPLES

We shall illustrate the use of the agent and cooperation frameworks proposed above by considering three examples, one from each of the cooperation strategies discussed earlier: one example from an ubiquitous travel service, one from image recognition and the final one from air-traffic control; the last two examples will be only cursory.

5.1 Travel Plan (Transaction-oriented Strategy)

Let us consider a simplified model of a travel service where there are a number of agents who cooperate together for booking plane tickets with hotel reservation and car hire. This example has four useful characteristics for our present purposes:

1. There is a need, at least in some queries, to exchange partial results.
2. The objects are heterogeneous.
3. The objects display overlapping, in some cases identical, capabilities.
4. The relationships between objects and skills are many to many.

We begin with a user model:

User Model

As stated earlier the user model has a task formulation model for the specification of tasks, and a response presentation model for response presentation, as shown below:

Task formulation model

```
Single (Cno, Source, Destination, Apdep, Quality)
where [condition (...)]

Return (Cno, Source, Destination, Apdep, Apret, Quality)
where [condition (...)]

Hotel (Cno,Area,Location,Asdate,Addate,Price-range,Unit,Rating,Quality)
where [condition (...)]

Car (Cno, Area, Location, Asdate, Addate, Model, Quality)
where [condition (...)]
```

Response Presentation Model

```
Single (Cno, Source, Destination, Carrier Flight, Depdate,
Deptime, Arrdate, Price Unit, Quality, Condition)

Return (Cno, Source, Destination, Carrier Flight, Depdate, Deptime, Arrdate,
Carrier Flight, Retdate, Rettime, Price Unit, Quality, Condition)

Hotel (Cno, Area, Location, Sdate, Ddate, Rating, Price Unit, Quality)

Car (Cno, Area, Location, Sdate, Ddate, Model, Price Unit, Quality)
```

We assume the item Quality holds a quality rating of the agent as perceived by this home agent.

Environment Model

The environment model consists of object classes, class-relationships, and function list (which includes all functions and procedures for invocation) The object classes are:

```
Airline  (Name, Area, Quality, Last-used, Last-changed)

Trag     (Name, Area, Quality, Last-used, Last-changed)

Bucket   (Name, Area, Quality, Last-used, Last-changed)

Tbureau  (Name, Area, Quality, Last-used, Last-changed)

Car-rentals (Name, Area, Quality, Last-used, Last-changed)
```

Trag stand for general high street travel agents, Bucket for cheap ticket agents and Tbureau for Tourist bureaus which specialise in hotel booking.

```
Class-link (Obname     Service    Last-used    Last-changed   Home)
             Airline   Single       -             -            -
             Airline   Return       -             -            -
             Airline   Hotel        -             -            -
             Airline   Car          -             -            -
             Bucket    Return       -             -            -
             Trag      Single       -             -            -
             Trag      Return       -             -            -
             Trag      Hotel        -             -            -
             Trag      Car          -             -            -
             Tbureau   Hotel        -             -            -

Regions (Area, Location, Last-used, Last-changed, Home)

Functions (Name, Tuple, Inverse, Parameters, Comments)
```

New or user-defined functions are added from the top and all functions are searched from the top, so that a user can over-ride an existing function by adding a new function of the same name. The Class-link relation lists skills of agent classes, Region yields location of places, and Functions lists library functions that are available.

Participation Model

Some examples of a participation model are:


```
Airline  (Name     Area      Quality     Last-used     Last-changed)
           BA      Europe    0.6
           BA      World     0.5
           BA      Germany   0.6
           LH      Germany   0.6
           AF      France    0.5

Trag     (Name     Area      Quality     Last-used     Last-changed)
           Cook    Belgium
           Cook    France
           Cook    Japan

Bucket   (Name     Area      Quality     Last-used     Last-changed)
           Stravel Europe
           UST     USA

Tbureau  (Name     Area      Quality     Last-used     Last-changed)
           Cook    Europe
           Amex    USA
           Ytour   Yugoslavia

Car-hire (Name     Area      Quality     Last-used     Last-changed)
           Alamo   Atlanta
           Budget  Atlanta
           Budget  London

Regions  (Area     Location  Last-used   Last-changed    Home)
           Africa  Nairobi
           France  Paris
           Europe  Brussels
           USA     Atlanta
           USSR    Moscow
```

Home Model and Communication Facility

The home model consists of the description of the home object, which is transparent to the global operations. For an airlines agent, such a model is likely to include items such as Start-city, Destination-city, Flight-no, Aircraft, Start-time, Destination-time, days and dates in which the flights are available, Price calculation procedure, etc. Since the specifications of the home model and communication facility are not needed to appreciate our presentation, we shall not discuss them any further here, except to point out that we assume the basic objects to be capable of performing their tasks.

Tasks

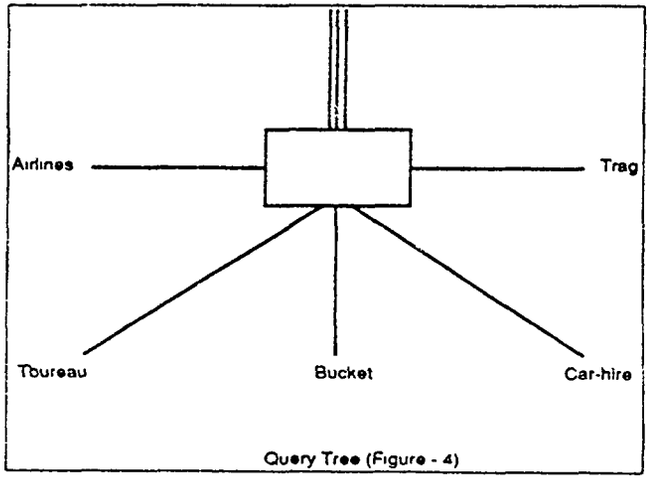
Consider the following user tasks

```
Return (Cno Source     Destination Apdep          Apret          Quality
Return [C10 London     Atlanta     Oct 12 ± 3     Oct 20 ± 3     > 0.4]
Where [Retdate - Depdate] = 8

Hotel   (Cno   Area   Location   Asdate   Addate   ...Rate
Hotel   [C10   USA    Atlanta    X        Y        ***]
Where [x = Return.Arrdate AND Y = (Return.Retdate-1)]

Car   (Cno   Area   Location   Asdate   Addate   %...)
Car   [C10   USA    Atlanta    X   Y
Where [X=Return.Arrdate AND Y = (Return.Retdate)]
```

Processing

This query is decomposed into three subtasks. Return, Hotel and Car, and the appropriate answer form produced. Converter for ratings and currency are also invoked These subtasks can be executed three separate ways.

1. By Airline alone
2. By Trag alone
3. By Tbureau, Bucket and Car-hire in cooperation, as shown in figure 4.

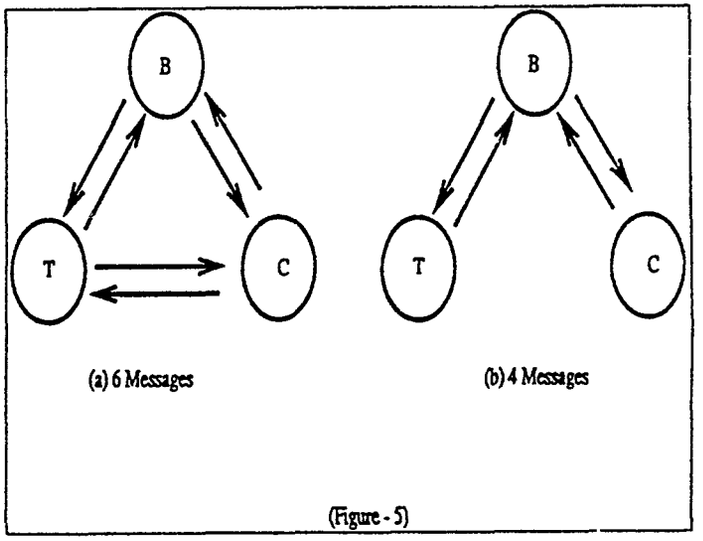


Query Tree (Figure - 4)

Assume we wish to find the minimum cost by exploring all three approaches. The Allocator will select suitable agents for the task/subtasks. The Airline and Trag agents will evaluate their response and return them to the requester node. For the third approach, which is more interesting to us, the agents must work in cooperation in two phases.

Phase (i): determination of the common dates for which flight, hotel and car are available.

Phase (ii): calculating the travelling costs for these common dates.



(Figure - 5)

In Phase (i) there are two ways in which the agents can communicate the common dates as shown in figure 5(a) and 5(b). In 5(a) each agent communicates with the others involving 6 messages, while in 5(b) T (Tbureau) and C (Car hire) communicate to B (Bucket), and the Bucket forwarding the final dates to T and C, with a total of 4 messages. We choose 5(b) to minimise the number of messages.

Once the Phase (i) is completed, the results are sent to the originating node. If any of the above agents are unable to perform the action, they can invoke others in turn, as explained earlier. The originating node will produce the final answer for minimum travelling cost, with help of the Lower Recomposer and appropriate converters.

If the task was for a World Tour rather than a return flight, then the environment model would have been updated, as it does not have an entry for World (Cno.....). If the Bucket

shop UST does not cover Atlanta, it will invoke its known acquaintances and if successful, the participation model of the requester node, will be updated accordingly. On the other hand if the flight is for a city in the USSR, the Learner will be invoked to update the participation model with new agents, since none of the flight agents in the present participation model for the requester node covers USSR.

5.2 Image Recognition (Recognition-oriented Strategy)

We assume a general case of a number of first level objects (nodes) which are capable of recognising patterns of interest from raw sensor data. The capabilities of the objects may vary, and therefore the first task in a recognition process is the selection of an appropriate set of objects. We also assume that there are a large number of sensors, some or all of them may be required in a recognition process; sensors may be dynamically linked to nodes and shared by nodes, since the nodes cover overlapping sensed-areas, to reduce the boundary effects. Each node sits at the top of a hierarchical set of objects (generally nondisjoint) with sensors lying at the lowest level, as in figure 6.

There are two possible ways of linking the nodes, as shown in figure 7(a) and (b); 7(a) is communication intensive, while 7(b) is application dependent. Unless there is an over-riding reason to the contrary, we advocate (b) as it is more efficient. We assume that wherever useful (at any level of the hierarchy), a superior object will be created for each application by the execution plan to reduce inter-object communications.

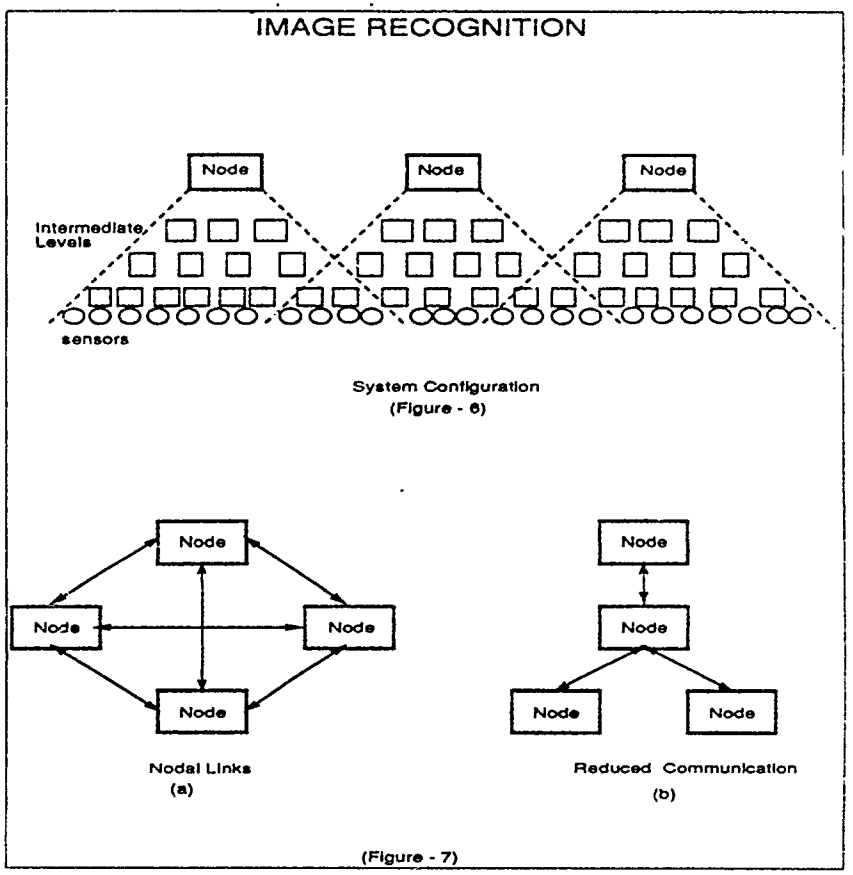
Image recognition is a bottom-up approach, since the raw data from sensors is processed upward leading to the recognition of the pattern. However, if we separate the preparation and execution stages, then we can generalise it with a top-down preparation phase backed by a bottom-up recognition phase. Furthermore, if we allow user-queries on recognised patterns, then we can treat this processing like a transaction model, where the pattern is a virtual item to be generated like a relational view facility. Thus, we can use the generalised transaction model given earlier. Consider the following skeleton for environment and participation models:

Environment Model

```
Patterns (name, identifier)

Object (pattern-id, node-id)

Nodes (node-id, pattern, reliability, coverage, version parameter)
```

Participation Model

```
Nodes (node-id, pattern, reliability, coverage, version parameter)
        N11      face     0.5          a12       3.6     ......
        N12      face     0.6          a12       3.7     .......
        N21      face     0.6          a13       2.1     .......
```

Task Processing Steps

1. The query from a pattern is validated against the expected structure and content of a pattern. The rest of the processing concerns the determination of the presence of a pattern.

2. The environmental model of the local node is used to identify the nodes of interest. If these nodes are not available, then the Learner will be invoked, thus leading to some updates of the environment and participation models. This stage will provide general

information for decomposing the area to be sensed (Higher Decomposer).

3. In the next level the participation models are invoked, and if they are out of date, then the foreign nodes concerned can be contacted for their updated versions. Nodes are selected along with a recomposition strategy (which might include integrators to resolve the boundary problems). The execution strategy will indicate which node will communicate with which other node(s).

4. Each node considers its part of the task and decomposes it further, again using some decomposition strategy. The process continues until the sensors (lowest level) are reached.

5. When all decompositions are completed, the executions can begin, following the plans prepared.

Given an object at any level of the hierarchy, it will treat the cooperation model of its lower-level objects as a black box; however, a subordinate object can contribute to more than one superior object, if so specified at the setup stage. The global coherence (resolution of boundary problems) will be achieved with appropriate integrators at each level; this we would view to be an application-dependent function, which we do not consider here.

5.3 Air-traffic Control (Control-oriented Strategy)

We shall select air-traffic as an example of control-oriented strategy. We consider a simplified model where the world is divided into regions, each region having a set of airports as shown in figure 8.

The regional controllers provide flight paths and exercise transit controls, while airport controllers supervise the take-offs and landings of planes. An appropriate authority requests for a flight plan which is passed to the local regional controller who consults with the controller of the destination region and those of the intervening regions, if any. Each regional controller in turn may have to consult the relevant airports. Once everyone agrees, the aircraft may take off. In a changeover from one region to another, there will be a standard protocol, which might require the aircraft (another agent) to change position, height, direction and speed.

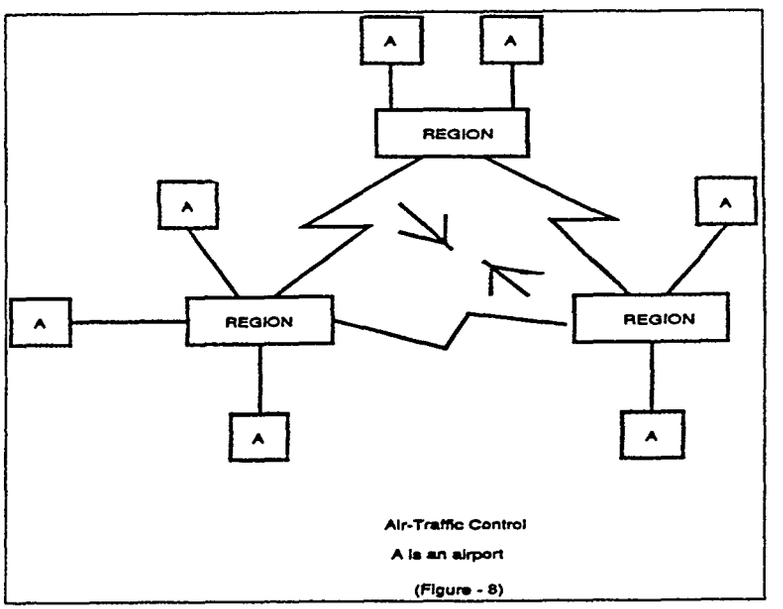
Skeletal environment and participation models will be:

Environment model

```
Region-map ( Source-reg,   Dest-reg,  Int-reg,    Quality)
             UK            Italy      France
             UK            Italy      Spain
             UK            USSR       France, Germany ...
```


Participation Model

```
Regports ( Region,    Airport,   Restriction, Communication point)
           UK         Heathrow
           UK         Manchester
           France     Paris
```



(Figure - 8)

Task Processing (Flying)

1. A requested route is passed to the Task Manager which invokes the Higher Decomposer to determine intervening regions on the route. The environment model of the local region is searched, and if the destination region is absent the learner is invoked, which may broadcast for assistance, resulting in the update of the environment and participation models. The process may lead to several sets of possible intervening regions. The Higher Decomposer invokes the appropriate structure for the result presentation.

2. The Lower Decomposer invokes the most convenient set of regions, following a decomposition strategy, extracts their contact addresses, and passes them to the proposed

schedule. If the answer is unfavorable, it may choose another set of intervening regions, again following a decomposition strategy. If the destination airport is busy, and if the original requester is happy with a nearby airport, the Decomposer will select one. The integrator will be invoked for sector-change protocols. If the process is successful, a route plan will then be ready. The success/failure is communicated to the requester. If the route is for a regular flight (equivalent to a repeated query), a route plan will probably be already in existence. In this case, the Task Manager will communicate to the appropriate regions for clearance. Once the clearance is given, it is communicated to the requester.

Then the execution phase begins:

3. When the plane takes off, the communicator communicates the message to all the relevant air-traffic controllers; but no time-out is probably needed.

4. The execution proceeds according to the execution plan. Action within each region will be the next level of activities within the regional controller, where controls will be exercised in accordance with the regional rules. These will be specified in the execution plans prepared by the Lower Decomposer, mentioned earlier.

CONCLUSION

We have introduced the concept of a Cooperating Knowledge Based System to provide multi-agent processing with a database flavour, emphasising three items: a general architecture, performance efficiency and concurrency/ reliability/ recovery facilities, of which we have discussed first two in detail. We have presented a generic model for the representation of agents, more generally encapsulated objects, backed by a hybrid cooperation model which is expected to provide efficient performance. We have also shown how these ideas can be used to develop applications in a wide range of application domains. By and large the work presented here is an exercise of applying some ideas of distributed databases on multi-agent systems.

Despite our attempt to produce generic framework, we must admit that the world of CKBS is far too complex to be solved by a single unified implementation. On the other hand, the idea of a tailor-made system for every application is also unacceptable in terms of development cost. The generic framework we have developed is based on an inherent similarity in the CKBS application areas; and this strengthens our belief that it should be possible to design some general-purpose systems, each of which can meet the needs of a whole class of applications. The proposed framework also provides scope for standardisation and formalisation, as in databases.

The proposed object model creates an autonomous object with an open interface, permitting not only heterogeneous but also pre-existing systems to act as an object in a cooperative environment. However, not all the facilities of the object model will be needed by each potential object. Likewise, the hybrid cooperation strategy provides a framework for many types of cooperation; a specific application may not need all the facilities. This framework

can also be used to describe further facilities such as concurrent usage, reliability, recovery, constraint propagation, reasoning and triggers. There are of course other issues, such as a high-level language for information exchange, which we have not addressed.

We are at the moment examining the following three applications, within the COSMOS project, which will use the ideas developed in this paper:

- Travel service
- Air-traffic control
- Network-traffic control

In due course, we also expect to improve the models in the light of our experience in these applications.

REFERENCES

[1] M. L. Brodie et al (editors): On conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer-Verlag, 1984.

[2] S. M. Deen and G.P. Thomas (editors): Data and Knowledge Base Integration, Proc. of a working conference, Pitman, 1990.

[3] A.H. Bond and L. Gasser: "An analysis of Problems and research in DAI", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 3-35 (This volume is strongly recommended to the new comers - this book reproduces important articles from past publications).

[4] E.H. Durfee et al: "Trends in Cooperative distributed problem solving", IEEE TKDE vol (1:1) pp 63-83, March 1989.

[5] L.D. Erman and V.R. Lesser: "A multi-level organisation for problem solving using many diverse cooperating sources of knowledge", Proc. International Joint Conf on AI, pp 483-490, 1975.

[6] E.H. Durfee, V.R. Lessor, D.D. Corkill: "Cooperation through Communications in a Distributed Problem Solving Network", Distributed Artificial Intelligence, edited by N. Huhns, Pitman 1987, pp 29-58. This book is also a recommended reading. A version of this work also appears on pp 268-284 in ref [3]. There are many papers on DVMT, we refer here only to some.

[7] E.H. Durfee and V.R. Lesser: "Negotiating task decomposition and allocation using partial global planning", Distributed Artificial Intelligence, vol II, edited by L. Gasser and M.N. Huhns, Pitman 1989, pp 229-244. This is another recommended volume.

[8] R. Davies and R.G. Smith: "Negotiation as a metaphor for distributed problem solving", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 331-356.

[9] R.G. Smith "The Contract Net protocol: high level communication and control in a distributed problem solver", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 357-366.

[10] G. Agha and C. Hewitt: "Concurrent programming using Actors ....", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 398-407.

[11] C.E. Hewitt: "Offices are open systems", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 321-330.

[12] R. Steeb et al: "Architectures for distributed air-traffic control" Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 90 - 101.

[13] S. Cammarata et al: "Strategies for cooperation in distributed problem solving", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 102-105.

[14] N.V. Findler and R. Lo: "An examination of distributed planning in the World of air-traffic control", Readings in Distributed Artificial Intelligence, edited by Bond and Gasser, Morgan Kaufman, 1988, pp 617-627.

[15] L. Gasser et al: "MACE: A flexible test bed for distributed AI", Distributed Artificial Intelligence, edited by N. Huhns, Pitman 1987, pp 119 - 152.

[16] R. Bisiani et al: "The Architecture of the Agora Environment", Distributed Artificial Intelligence, edited by N. Huhns, Pitman 1987, pp 99-118.

[17] R. Engelmore and T. Morgan (editors): Blackboard systems, Addison-Wesley 1988. This edited volume contains a large number of systems that use the blackboard architecture.

[18] M. Kamel and A. Syed: "An object-oriented multiple agent planning system", Distributed Artificial Intelligence, vol II, edited by L. Gasser and M.N. Huhns, Pitman 1989, pp 259-290.

[19] A.H. Bond: "The cooperation of experts in engineering design", Distributed Artificial Intelligence, vol II, edited by L. Gasser and M.N. Huhns, Pitman 1989, pp 463-486.

[20] S. Ceri and G. Pelagatti: Distributed Databases - Principles and Systems, McGraw-Hill, 1985.

[21] S.M. Deen et al: "Data Integration in Distributed Databases", IEEE Transactions on SE, Vol (SE 13:7), July 1987, pp 860-864.

[22] S.M.Deen et al: "The architecture of a distributed database system - PRECI*", Computer Journal, Vol (28:3), Aug 1985, pp 282-290.

[23] S. Bhalla et al: "A technical comparison of heterogeneous distributed database management system," MIT technical report 1987 (copies from Dr. A. Gupta, Sloan School of Management, MIT, Cambridge, Mass 02139). This report reviews a number of major distributed database systems.

[24] S.M. Deen et al: "Distributed directory database system for telecommunication", Computer Journal, Vol (31:12), April 1988, pp 175-81.

[25] W. Litwin and A. Abdellatif: "Multidatabase interoperability", IEEE Computer Vol (19:12), Dec, 1986, pp 10-18.

[26] U. Dayal: "Query processing in multidatabase system", Query Processing in Database System, Springer-Verlag, 1985, pp 81-108.

[27] J.R. Ensor, J. D. Gable: "Transactional Blackboard", Blackboard Systems, R. Englemore and T. Morgan (editors), Addison-Wesley, 1988, pp 465-474. A recommended book to read.

[28] B. Hayes-Roth and M. Hewett: "BB1: An implementation of the Blackboard Control Architecture", Blackboard Systems, R. Englemore and T. Morgan (editors), Addison-Wesley, 1988, pp 297-315. The same authors have another article on BB* in the same book on pp543-560.

[29] D.D. Corkill et al: "GBB : A Generic Blackboard Development System", Blackboard Systems, R. Englemore and T. Morgan (editors), Addison-Wesley, 1988, pp 503-516.

[30] D.M. Lane, A.W. Quinn, International Conference on CKBS (this conference), 1990.

A 22

Meeting the Cooperative Problem Solving Challenge: A Database-Oriented Approach

S. Chakravarthy[1] S. B. Navathe K. Karlapalem A. Tanaka[2]
Computer and Information Sciences Department
Database Systems Research and Development Center
University of Florida, Gainesville, FL 32611, USA

Abstract

(Abstract for the whole paper)

Cooperative problem solving is a complex activity requiring harmonious interaction between active agents (typically humans providing sequencing, decision making, and coordination components) and systems (typically passive - providing inferencing as well as algorithmic computation). Each component (or problem-solving node)is capable of sophisticated problem solving and can work independently, but the problem faced by the entire set of components cannot be completed without cooperation. Several problems addressed in the literature, such as automation of office environments can be viewed as a special (and perhaps a simple one) case of cooperative problem solving. In our view, recent advances in DBMS technology (e.g., active and object-oriented DBMSs) and maturation of other concepts (e.g., temporal databases) provide us with techniques and abstractions for formulating a viable solution to the above problem.

In this paper, we analyze the problem of cooperative problem solving and identify key underlying characteristics. We then propose a database-oriented solution by combining and extending techniques and abstractions to support the characteristics identified. Specifically, we draw upon techniques from the areas of database management, artificial intelligence and knowledge based systems. We also describe viable immediate and long-term solutions for supporting cooperative problem solving. Finally, we elaborate on critical aspects of our multi-staged solution.

[1]Parially supported y the Florida High Technology and Industrial Council (UPN# 89090848)
[2]Supported by the Brasilian Army and CNPq - National Research Council (RN 201108/88.7)

Extended Abstract of the Paper

1 Characteristics of Cooperative Problem Solving Environments

A large number of activities (including problem solving) carried out by humans requires synergistic collaboration between humans (i.e., active agents) and systems (i.e., passive agents, typically). Each component (or problem-solving node)is capable of sophisticated problem solving and can work independently, but the problem faced by the entire set of components cannot be completed without cooperation. These *collaborative*[3] *activities* also involve extensive access to information repositories, and computations. The salient characteristics of *collaborative activities* are: *decision making* using a set of criteria (e.g., selecting candidates to be called for interview, selecting the computing environment for a project), *computation* using a variety of systems and tools (e.g., searching a database to select referees, computing the optimum layout of components in circuit design), and more importantly *collaboration* (e.g., dialogue, communication, mediation and specifically, reaching consensus, agreeing upon the candidates to be called for interview from the selected list). In order to support collaborative activities with these characteristics, it is imperative that the underlying computing environment be capable of providing support for all of the characteristics identified.

Many day-to-day activities in the world fall into the above category. For example, search for a new faculty member (or any personnel as a matter of fact) involves collaboration (among the existing faculty), decision making, as well as computation. Perhaps diagnosis of an ailment involving multiple autonomous medical experts exemplify the problem of cooperative problem solving and all of its facets. Finally, office automation, a specific example of the collaborative work environment, also exhibits all the characteristics mentioned above. In order to underscore the complexity of these class of activities, it is useful to contrast them with conventional activities carried out using extant computing environments and tools. Most of the activities fall into one of the following two categories: *computation-intensive*, where there is very little interaction with the outside agent (e.g., wind-tunnel simulation, finite element analysis) and *one-on-one interaction* with an agent and the system (e.g., interactive applications).

On the other hand, a collaborative activity requires considerable coordination among a

[3]In this paper, we use the words collaboration and cooperation interchangeably

large number of agents as well as systems and agents. Currently, a significant part of this coordination – both among systems, and systems and agents – is done by the (active) agents. As a consequence, agents are an integral part of the loop performing required activities that can be automated to a large extent. Our goal is to suitably enhance the functionality of the underlying components to support the execution of collaborative activities. Our approach first identifies appropriate techniques and abstractions from several relevant technology areas and then describes how they can be adapted and combined to provide the required functionality in the underlying computing environment. In order to provide better support for the specification, management, and execution of collaborative activities, it is imperative to raise the level of abstraction at which collaborative activities are specified to the environment. This in turn enhances the functionality of the environment to accept, execute, and manage higher-level specifications.

We believe that the database management technology can be suitably enhanced to provide the functionality required for collaborative problem solving. Currently, DBMS research is identifying and proposing solutions to meet the requirements of non-traditional applications. In addition, a number of useful techniques developed in the areas of software engineering and AI (specifically, expert systems, and to some extent coordinated distributed problem solving [BARR89]) are also relevant. However, it should be pointed out that none of the technologies, individually, is adequate to support the problem addressed in this paper. Hence the need for an eclectic solution.

2 Overview of our approach

In this section, we provide an overview of our approach indicating: i) the relevance and usefulness of techniques culled from relevant technologies and ii) an evolutionary path leading to an integrated system for supporting collaborative problem solving.

An analysis of the characteristics of a collaborative activities suggests the need for the following functionality in any environment supporting collaboration:

1. ability to model complex (collaborative) activity,
2. ability to support (rule based) inferencing for decision making,
3. ability to maintain historical information pertaining to an activity for: tracing its progress, querying, and inferencing,

4. ability to maintain temporal information pertaining to an activity to perform sequencing, coordination as well as inferencing (over temporal data),
5. ability to communicate asynchronously (perhaps using alerters, triggers, and other mechanisms such as mailbox) with other systems as well as agents,
6. ability to capture the capabilities of the constituents (heterogeneous systems and agents) as part of the data/knowledge base, and
7. ability to maintain and manage a data/knowledge base shared by the activities (in terms of consistency and being able to recover from failures).
8. ability to solve problems in parallel and build compatible solutions

Techniques for modeling data and processes will be combined with relevant methodologies from software engineering to model collaborative activities as well as active and temporal components of the data/knowledge base (item 1). Amalgamation of knowledge-representation techniques [BRAC80] into the data model of DBMSs brings with it the concomitant techniques developed for AI systems. For example, hypothetical reasoning can be used for supporting "what-if" scenarios. As another example, rule-based inferencing and explanation of conclusions arrived at can also be realized in a straightforward manner. Also, rule-based inferencing needs to be extended to include temporal data (items 2 and 4)

One of the key components of our approach is the active database abstraction which will support the coordination of activities among systems as well as systems and agents. An active DBMS supports events (database, non-database, temporal, and composite) that can be specified on the data stored in the data/knowledge base (refer to Figure 2.2.) (item 5). Another key component of our approach is the support for maintaining both historical and time-stamp based data. These will be supported by the underlying DBMS used for the purposes of coordination (items 3, 5, 6, and 7). Loosely connected network configuration accommodates parallel problem solving and compatibility of interdependent solutions can be ensured by the use of mechanisms such as the blackboard (item 8).

2.1 Multi-staged Approach to Cooperative Problem Solving

In this subsection, we first discuss the conventional approach currently being used for collaborative problem solving and its limitations. We then propose our approach and contrast that

with the conventional approach. Finally, we provide a long-term solution which combines most of the concepts that will be developed as part of this work into an integrated solution.

Figure 2.1 indicates the current approach to collaborative problem solving. The main drawback of this approach is that the agents (usually one or more people engaged in collaborative problem solving) have to provide most of the coordination among agents as well as among systems and agents. Systems and tools are used as passive components capable only of performing computations. The onus of information propagation, reminding other systems/agents, and sequencing of activities, checking for compatibility of solutions etc, are on the agents involved in problem solving. Any system specific capability (such as e-mail, calendars etc.) is exploited in an ad hoc manner to solve the problem. The problem being solved cannot be specified to the system at a conceptual level that includes all the coordination, sequencing, and dataflow information.

Our approach (Figure 2.2) is based upon the pragmatic assumption that the systems participating in a collaborative activity are autonomous and the execution of an activity can be coordinated by using a combination of database techniques developed for satisfying the requirements of non traditional applications. An active, temporal knowledge base (ATKB) in conjunction with an active blackboard (AB) is proposed as a mechanism for accomplishing cooperation among problem solving nodes (including active agents). An activity coordinator (which is analogous to a transaction manager of a DBMS, as shown in Figure 2.2) is introduced for the purpose of coordinating the components of an activity. The combination of the active and temporal capabilities will provide the requisite support for the coordination of component activities among the constituent systems over which they are defined. Any lack of inferencing capability at any of the systems will be compensated by the functionality of the active, temporal knowledge base management system. The only assumption used in our approach is that the activity coordinator be able to communicate and exchange data in some manner with other problem-solving nodes (systems and agents) that are part of the network. A collaborative activity will be specified to the system in a declarative manner which will be translated into metadata and triggers/alerters on that metadata in the ATKB. The blackboard will be accessible (for read and write) by all the nodes in the network. Where necessary, the blackboard will be used as a globally accessible space for posting solutions to subproblems to have their compatibility checked. The actual execution of an activity (including coordination and sequencing) will be handled by the activity coordinator which will be responsible for information propagation, sequencing, and coordinating subactivities

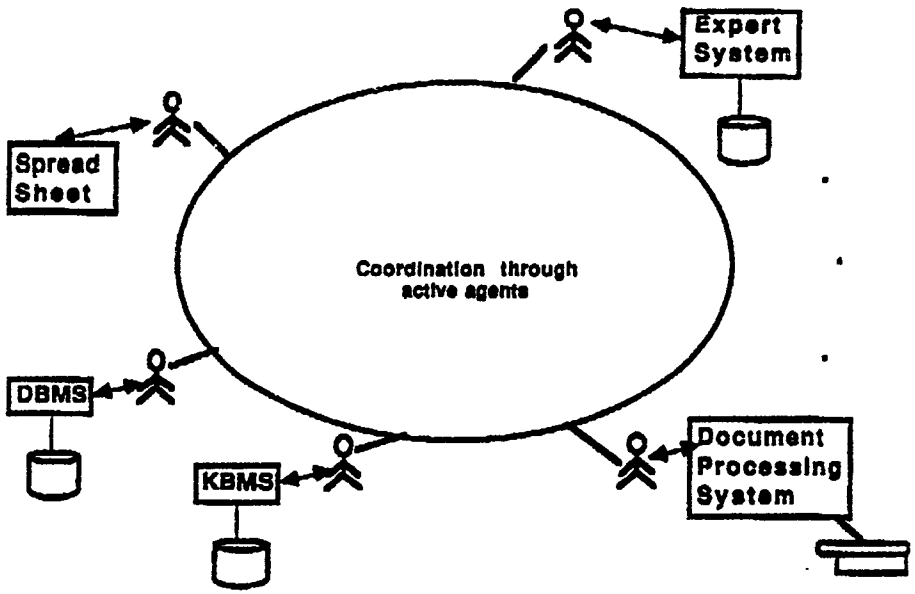


Figure 2.1 Conventional cooperative problem solving environment

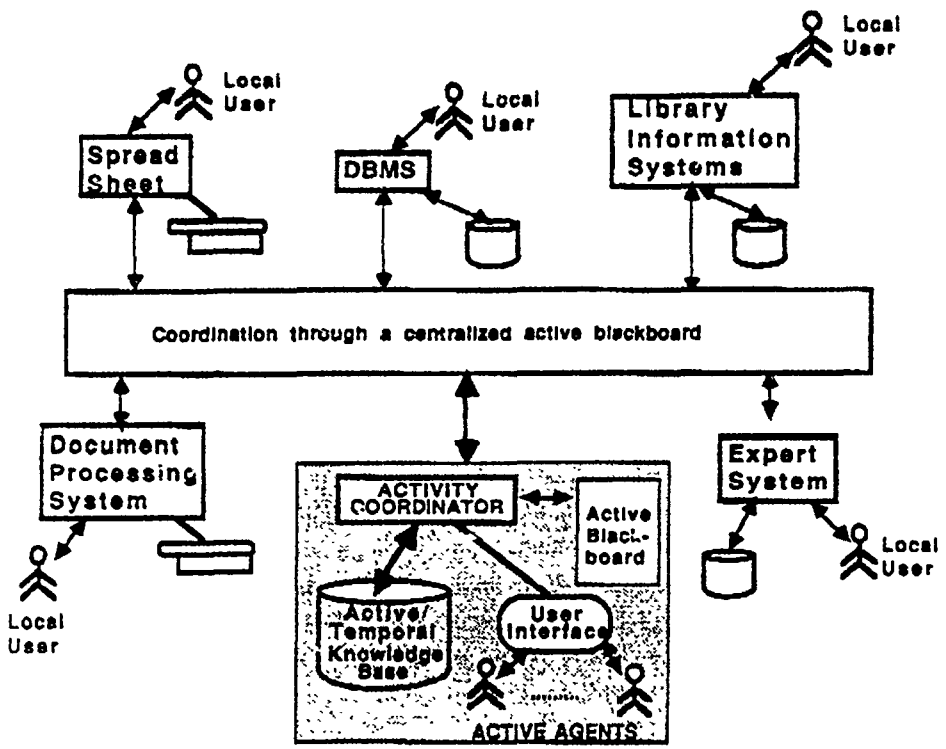


Figure 2.2 Database-supported cooperative problem solving environment

4b

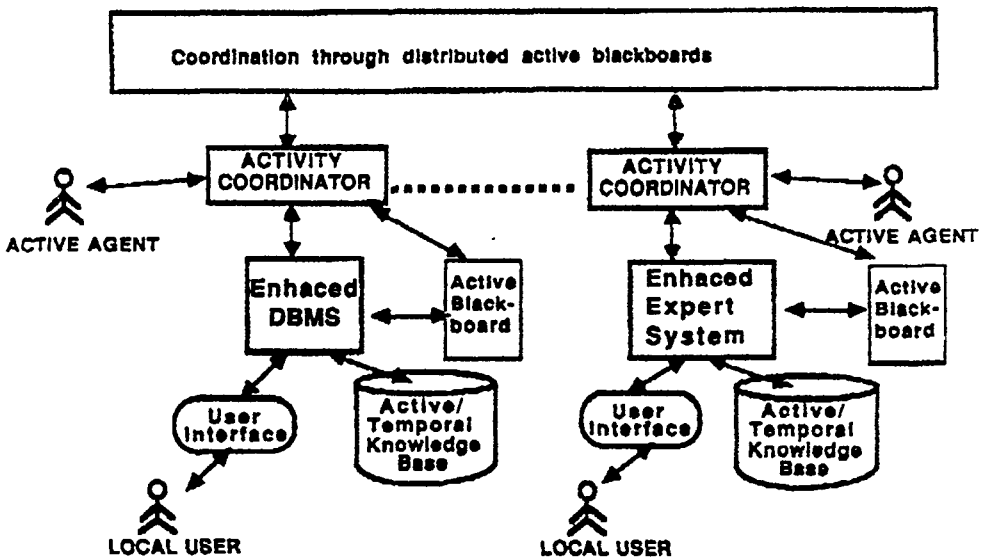


Figure 2.3 Integrated cooperative problem solving

4c

and atomic activities

The essence of our approach is that the user can now specify a collaborative activity conceptually and the onus of its execution (optimization, management, interference with other activities) is on the activity coordinator and the ATKBMS. The activity coordinator, in conjunction with the active, temporal knowledge base and AB will execute the activity requesting inputs from appropriate components (be it passive systems or active agents). Agents involved in the problem solving activity will play only a supervisory role in providing their inputs or decisions in contrast to the situation shown in Figure 2.2.

Figure 2.3 illustrates the next generation of collaborative problem solving environment. Ideally, each component of the environment should be capable of supporting collaboration which is possible only if each of them is extended (overriding the autonomy of the system) or replaced with the initial system augmented by the capabilities of the activity coordinator and the underlying knowledge base as shown in Figure 2.2. Hence, we propose this as an integrated solution into which results obtained from this effort need to be incorporated. Also, the autonomy assumption cannot be made in this case and as a consequence, a migration path from existing systems need to be outlined. It is more likely that, in the long-term, we will have a combination of the two (some autonomous and some integrated systems) rather than the combination shown in Figure 2.3. Hence, the long-term solution needs to address merging of the architectures proposed in Figures 2.2 and 2.3.

Finally, a clarification about the proposed database-oriented approach. Unlike the AI approach to modeling cooperative distributed problem (CDPS) solving [BARR89] in terms of negotiation, functionally accurate structuring, multi-agent planning etc., we take a more pragmatic (and a bottom up) approach in terms of defining and solving the problem with known and viable DBMS functionality. Although the AI approach is desirable in the long-term. a better handle on the problem cab be obtained using the proposed approach.

3 Relevant Technologies

Although a fair amount of research has been done in modeling and supporting office environments, recent work in active, temporal, and object-oriented databases warrant a fresh look at the overall problem of collaborative work environments. Below we review literature in areas identified as relevant to the problem addressed in this proposal.

3.1 Active Database Management

Active databases can be viewed as an abstraction for elegantly supporting a variety of database functions that were being realized using special-purpose mechanisms in conventional DBMSs. Traditionally DBMSs have been passive; that is queries or transactions are executed only when explicitly requested.

The key advantages offered by the active database abstraction are modularity and timely response. Modularity is achieved by separating the conditions (and actions) to be evaluated from the application code as well as relieving the application programmer from having to specify the periodicity with which to check whether the conditions have become true. Timely response is obtained by virtue of the system evaluating the conditions in the most appropriate manner without having to wait for the next polling cycle. [CHAK89a] point out the usefulness of rule-based active capability for supporting a variety of DBMS functions, such as integrity and security enforcement, access control, maintenance of derived data, materialized views and snapshots, and rule-based inferencing.

The asynchronous execution of triggers and alerters in actives databases provides the basis for problem dependent asynchronous communication among the problem-solving nodes. The blackboard acts as an active global scratch pad to which partial solutions can be posted by any node and referred to by the other nodes.

3.2 Temporal Databases

Time plays a central role in collaborative work environments. For example, in the faculty search problem, it is important to keep track of the availability of the candidate for interviewing, the order in which final offers are made and even the scheduling of interviews relative to the availability of core faculty in that area. As another example, administrative and legal information systems must record changes in policies, procedures, and laws over a long time span and must consistently apply them to events taking place over the time span.

Recently, numerous researchers have addressed the problem of incorporating the semantics of time into information systems. Surveys appear in [BOLO82, DAYA85, SNOD86]. Most of the work reported on temporal databases extend the relational model for time support. Special time attributes are added to flat relations by [JONE80, BENZ82, CLIF83, LUM84, ARIA86, NAVA89, SNOD87], which is equivalent to time-stamping of *tuples*. Another way to classify the different proposals is by using the concepts of valid time and

transaction time dimensions [SNOD85]. Valid time leads to historical databases, while transaction time leads to rollback databases. Other research in temporal databases includes [ANDE81, CAST82, GINS84, ADIB85, SHOS86, NAVA87, SARD87, SEGE88]. Little work has been done on conceptual modeling for temporal database applications barring [CHEN86, ELMA89, THEO90], still restricted to the valid time dimension. In our work, we propose to provide temporal modeling at both conceptual and logical modeling level. Both valid and transaction time dimensions, as well as the mapping from the conceptual to the logical schema are important to our approach.

Combining active and temporal abstractions provides a powerful mechanism for asynchronous communication which can either be based on events corresponding to arbitrary computations or time or both.

3.2.1 Office Information Systems (OIS) and Cooperative Work

The characteristics of the office information systems are similar in nature to those found in the collaborative environments. In fact, an office information system is an ideal case study to illustrate our approach.

Many conceptual modeling and design methodologies for OIS have been proposed [SIRB81, DUMA82, KONS82, LOCH88, MAZE88, PERN88, PERN89]. Also, a lot of work has been done on supporting office work at progressively higher levels, by using data-based models [WHAN87], AI techniques [RIEG81, BARB83, CROF84, KAYE87], data semantics and procedures [ATTA82, GIBB83], concurrent computation in distributed systems [AGHA87], and message systems [CORT84, MAZE84, KOO88]. Recent work on CSCW includes [GREI87, MALO87, STEF87, SAMA88].

Our approach similarly explores the role of computers in managing activities performed by cooperative, coordinated, and collaborative systems under various temporal and non-temporal constraints. The concept of the activity coordinator differs from that of a mediator[WIED89]. While the mediators are to contain the administrative and technical knowledge to create information needed for decision making, the activity coordinator uses the information managed by the active, temporal knowledge base to take decisions. The decisions are taken as to how and when the activity is to be executed is made by detecting events and initiating actions.

4 Discussion of Our Approach

In this section we elaborate on the architecture proposed in Section 2 and give a brief description of the components. The environment consists of a loosely coupled network of problem-solving nodes. Duplication of functionality among the nodes is permitted. The goal is to perform a set of collaborative activities that are defined to the environment. Figure 4.1 shows the details of the approach shown in Figure 2.2. The components of the environment are a set of nodes, an activity coordinator which coordinates the set of activities, and an active/ temporal knowledge base system (ATKBMS) and the corresponding knowledge base, ATKB, store information about activities used by the activity coordinator.

We briefly describe some of the concepts and the functionality of the components shown in Figure 4.1:

Nodes

Nodes are those components that perform the sub-activities. A human can act as a node having a distinct set of roles, responsibilities, constraints and functions. A node, in general, is be a collection of programs providing expertise, resources, and information to solve a subproblem. Nodes are typically static and the specification of their structure and functionality does not change over long periods of time. Hence, these nodes do not evolve but they tend to be replaced. This replacement is done mostly to improve the functionality and/or the capability of the system.

Activities

Any problem to be solved collaboratively is modeled as an activity. An activity in general consists of a set of sub-activities. An activity is executed over long periods in contrast to typical database transactions that have a short duration. An activity, to be executed, requires a subset of the resources of the network and more importantly a schedule that involves communication among the nodes at either prespecified points in time (temporal) or asynchronously (data-flow like) or a combination thereof.

Activity Coordinator

The activity coordinator is responsible for the execution of an activity in terms of its subactivities. It is also responsible for matching the requirements of subactivities with the resources available in the network. It uses the scheduler and the blackboard

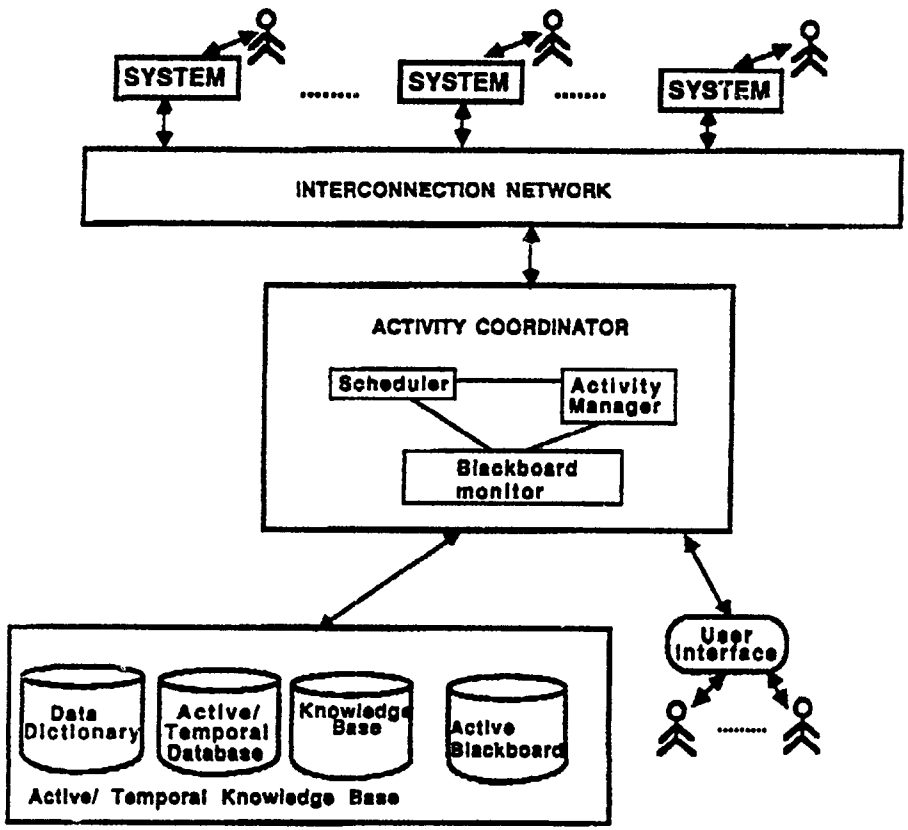


Figure 4.1 Details of the Proposed Architecture

monitor to kee track of the progress of an activity and how various subactivities are schedule in the network.

Interaction between Activities and Nodes

The functionality that is required to perform an activity is mapped to the capabilities of the nodes and a plan is generated. This plan is generated by a component of the environment known as the Activity Coordinator. This plan is the model of the interaction that is required within the nodes, within the activities, and between the activities and the nodes This plan defines the protocol that is used by the activity coordinator in inducing the collaboration within the nodes. It is transparent to the end-user. This collaboration is explicitly generated by means of events and interrupts that are defined by the activity coordinator and managed by the ATKB using the blackboard.

Active, Temporal knowledge base

The specification of activities and interactions defines the structure of the active/ temporal database in order to support the plan for managing the set of activities in the environment. An active database system supports events and interrupt mechanisms, by monitoring the database. An active, temporal database has the capability of monitoring temporal events and initiating a set of actions when certain situations arise. The plan generated by the activity coordinator defines the set of situations that can arise while processing a set of activities and the set of actions that need to be taken as a result of those situations. These situations and actions are modeled in terms of the schema of an active, temporal database. At run time, this schema gets populated. The knowledge base is used to store the knowledge regarding the extensional and intensional data of the nodes and environment. Hence a methodology for implementing an active temporal database design tool forms a part of IICE. The active, temporal capability of a database plays a very critical role in this environment.

Blackboard

The blackboard acts as the global scratch pad for posting and accessing any information for performing an activity. Both data used for controlling the activity as well as partial results can be posted to the blackboard. The active capability facilitates of the blackboard facilitates the coordinator to separate meta data required for coordinating

an activity with other types of data (e.g., capability and resource availability at each node).

An example and other details will be included here

5 Conclusions

In this paper, We have mapped the requirements of cooperative distributed problem solving environments on to the functional as well as architectural extensions of conventional databases. We Identified the relevant technology areas, briefly described them, and established the utility of combining techniques from seemingly diverse areas. We have proposed the notion of an activity as an underlying mechanism for modeling, specifying, and executing problems that require cooperation.

We have proposed two approaches for collaborative environments – one immediate tern and one long term – taking into consideration the feasibility and viability of the components involved. For one of the architectures, we have presented the functional components, interactions among the components, and interactions between the components and the rest of the system.

We have described our approach for executing an activity by means of a collaborative schedule and illustrated it by means of an example. Currently, we are developing methodologies for modeling, languages for logical specification, and algorithms for executing activities using the approach presented in this paper.

6. References will follow

Intelligent Agents in Federated Expert Systems

-

Concepts and Implementation

St. Kirn, G. Schlageter

University of Hagen
Department of Computer Science I
P.O.B. 940

D-5800 Hagen

Abstract

Today's expert systems lack possibilities to solve great, complex and heterogeneous tasks. One approach to handle these problems deals with the development of cooperating systems. Because of the major characteristics of the agents in such an environment (autonomous decentralized, independent) we name the global system "federative expert system". Federative cooperation of intelligent systems leads to a broad range of interesting questions. Some of these are local and global architectures, estimating system's ability to solve given problems, task allocation, task decomposition and synthesis of (sub-) results.

In this paper we describe a concept of federative cooperation between intelligent agents and discuss the major features of such systems, which are to be integrated in an overall system of cooperating agents. Finally we present an implementation of a federative expert system, which solves complex and heterogenous tasks in a banking application.

1. Introduction

The transfer of today's AI technology from research projects to industrial use has shown a variety of problems. Some of them deal with the need of very special hard- and software-resources, some other with the acceptance of AI-systems by the end-users, but a lot of problems result from central aspects of modelling and representing knowledge. First of all, these problems occur with the design of great, complex and heterogeneous applications ([Wied 89]), because of

1. problems with the administration of large knowledge bases:

- maintenance and consistency of knowledge,
- only one domain in a knowledge base: the number of relations between the rules in a knowledge base grows faster than the number of rules itself,
- more than one domain in a knowledge base: relations between different domains require additional administration resources

2. problems during use of great knowledge bases:

- performance of problem solving and
- validation of results.

One approach to deal these problems for design and practical use of today's expert systems are strategies of cooperative problem solving. From cooperative problem solving we expect the following essential advantages

- extensibility of the overall system
- maintenance of knowledge bases and
- performance of problem solving.

Federative Expert Systems

Along with [Deen 88], [ScKl 88] we name a system of cooperating expert systems federative, when it shows the following characteristics:

- the local systems are autonomous,
- they are loosely coupled (for example via blackboard architectures or contract net),
- they initiate cooperation only, if they identify a need for cooperative problem solving,
- they take part in a cooperation only, if they decide, that they are able to support the global problem solving process.

Today's research efforts in Distributed Artificial Intelligence deal mostly with those aspects, which must be solved in the overall system, like inter-agent communication, cooperative problem solving among multiple experts, intelligent coordination of cooperative problem solving and so on (see [Bond 88]).
In this paper, we turn our view to the question how intelligent agents take part in cooperation processes and what are the properties needed, if agents will be integrated in federative expert systems.

The paper is organized as follows: In chapter two we start with an example, which gives an overview over federative problem solving processes. This example helps to identify important properties of cooperating intelligent agents, too. On this basis, chapter three develops a general concept of local behavior during processes of federative problem solving. Then, chapter four discusses in detail the key features of intelligent agents, which take part in federative cooperation processes. Finally, chapter five introduces an implementation of a federative expert system, implemented under [illegible] which [illegible] problem [illegible] banking [illegible]

2. Federative Problem Solving: An Example

In this section we describe a federated expert system, which contains five intelligent agents. Each of the agents is able to solve special problems in the area of mathematics. The problem solving capabilities of the five agents are defined as follows.

- expert system A: polynomial calculus
- expert system B: differential calculus
- expert system C: integral calculus
- expert system D: trigonometrical calculus
- expert system E: power calculus

The inter-agent communication, task description, definition and decomposition is based on the context-free grammar G, which is defined as follows.

G := ({E, T, F}, {O, V, C}, R, E)

with a rule set R:

E --> E+T | E-T | T | INT E | ABL E | FAK E | ABS E | POT E | sin E | cos E | tan E | cot E
*T --> T*F | T F | F*
F --> (E) | a | b | c | x | y | z

and

E: expression (and starting symbol)
T: term
F: factor
V: set of variable names, defined by V := {x, y, z}
C: set of constants, defined by C := {a, b, c}
O: set of operators, defined by

*O = {+, -, *, /, INT, ABL, FAK, POT, ABS, ( ), sin, cos, tan, cot}.*

The language L(G) can be used to define rather complex mathematical tasks. For example, the following term can be derived from G by a simple left side derivation

$$f(x,y) := \frac{INT(1/(\cos^2(P(x)/Q(x))-a^2)^{0,5}) + INT(1/(a^2-x^2)^{0,5})}{ABL(\arcsin(Q(y)/a))}$$

With G, this term can be described as a syntax graph, which itself gives one possibility for task decomposition. The root of the graph defines the overall task, each sub-node represents a sub-task (Fig. 2.-1)

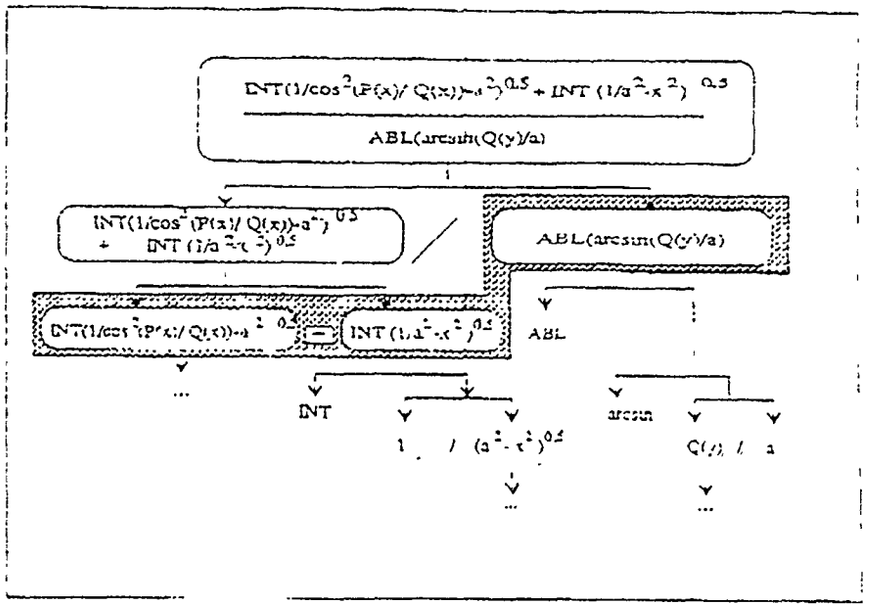


Fig. 2.-1: Syntax-based task decomposition

Now we look at the global process of cooperative problem solving. In our example we give the overall task to agent A. A isn't able to solve this task, but it will continue problem solving with task decomposition. The resulting subtasks may be those marked in Fig. 2.-1. Then agent A sends descriptions of the three subtasks to agents B, C, D and E. These estimate there abilities to solve those tasks. An essential point here is, that the agents shouldn't start local inferences to estimate their competence. Instead of this, they estimate their competence by comparing abstract descriptions of tasks with their own (local) problem solving capabilities.
As result of estimating the competence of the five agents, problem solving will continue with the agents B and C. Now, both agents continue with local problem solving. Again, each of them will reach situations, which require further cooperation. After finishing local problem solving, each agent returns its local results to its customer, which synthezises the sub-results in his (local) overall result. Fig. 2-2 describes one possible communication structure. This structure depends primarly on the starting point of the problem solving process (and sub-processes), the decisions to decompose a given task in subtasks and the choice of agents to solve them.
A general description of federative problem solving processes looks as follows. The first step is to decide, which of the agents should lead the overall problem solving activities. In a second step, this agent has to define that degree of task decomposition, which it finds usable to solve the given

problem. The third step deals with the allocation of sub-tasks to competent local systems, which are responsible for solving these subtasks. This process is cyclic, it continues over several levels of task decomposition and task allocation. After all, sub-results reached within local inferences are returned to the customer, which integrates them in his own results; a process, which continues, until the overall task can be returned to the user or subtasks couldn't be decomposed further.

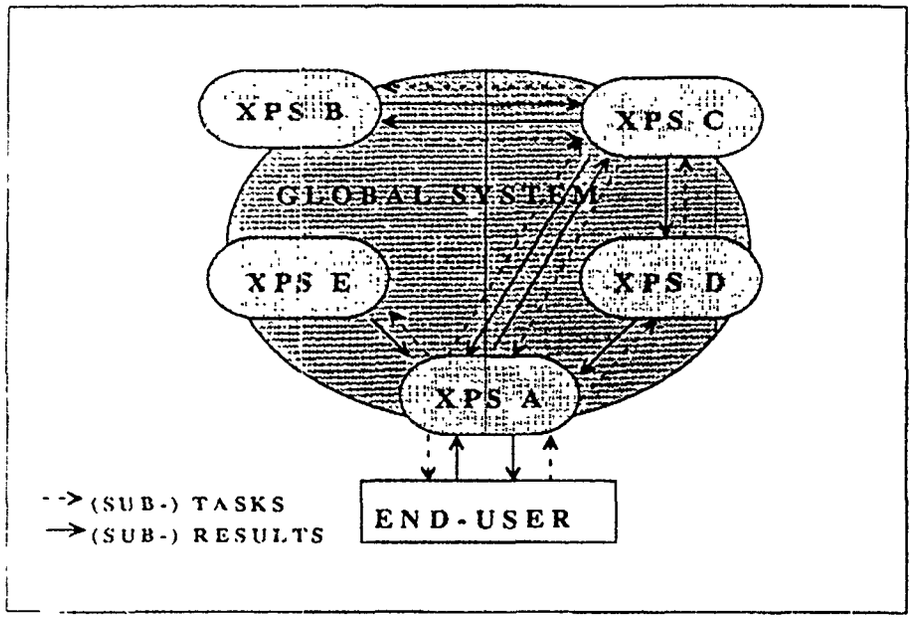


Fig. 2.-2: Dynamic cooperation in federated expert systems

What do we learn from this example?

Even without analyzing all details we can see from this example the essential characteristics of federative expert systems. First of all, the interacting agents need some common internode language. At least, this language serves as basis for inter-agent communication (in general that means communication between heterogeneous agents), task description, task decomposition and synthesis of results.
Then, task decomposition could lead to more than one possible set of subtasks, even, if there exists only one syntax graph of the task. In reality, an agent could have different possibilities to decompose a given task. That leads to further decentralization of the dynamic inter-agent problem solving process. That means, we have to find criteria to decide, which of them is the best one to prefer against the others.
Third, task allocation occurs, which is based on negotiation processes between the agents. The allocation of a set of tasks to a group of intelligent agents requires some additional features of cooperating expert systems, which are discussed below.
Finally, the agents must be able to synthesize sub-results to an overall result.
We see a rather interesting result in the fact, that the expert systems don't need to have any local knowledge, what problem solving capabilities are given and how they are distributed in the overall system. This feature gives a great chance to change the global configuration of the federative expert system, especially for maintenance dynamically, to change problem solving capabilities of local agents, to remove old agents from or insert new agents into the federative expert system.

3. How Agents Act in Federative Problem Solving Processes

In this chapter we describe a simple model, how intelligent agents act during the process of federative problem solving. The model itself provides a basis to identify those key problems of federative problem solving, which will be discussed later in chapter 4.

Like in Smith's contract net system we call an agent, which distribute tasks the manager of those tasks. Those agents, which accept subtasks from managers are called contractors. The overall systems architecture and the global process of problem solving is designed as contract net, which we don't outline here (see, for example, [Smit 80], [Klet 89]).

As we have seen, in a federative expert system every agent can get tasks, which can't be solved by it. That means, in a first step the agent has to check, if the given job can be done locally. This is a completely new step in expert system problem solving. In some cases, this check will result in an answer, which says, that the agent can do the job definitly, or definitly not. In most cases however, the result will be, that there exists no definitive possibility to answer this question. If it is able to do the job, local inference starts. If it isn't able to do the task, it will try to decompose the task to continue local problem solving. If decomposition of the task isn't possible, too, it will return it to the task's manager.

Decomposition of tasks could result in the widely known problem of information loss. That requires to identify lost informations and to add them to the received subtasks. Then these subtasks are described on an abstract level to prepare the task allocation process.

The allocation of sub-tasks has to be discussed in more detail. Tasks can be allocated only, if a system knows, which agent is able to solve this task. But in which way an agent could receive this knowledge? Three basic possibilities are given:

- All agents know, which problem solving possibilities are given in the overall system and how they are distributed in the system.
- Only one agent possesses this knowledge. It acts as a supervisor and is responsible for global task allocation.
- None of the systems possess such global knowledge; if they want to distribute tasks, they start negotiation processes, which result in a concrete (sub-) task allocation.

As we have shown in chapter two, federative expert systems tend to use the third alternative. If one agent wants to distribute a given task, it receives the needed information by broadcasting a task description. The other agents estimate their own capabilities to solve this task and return their

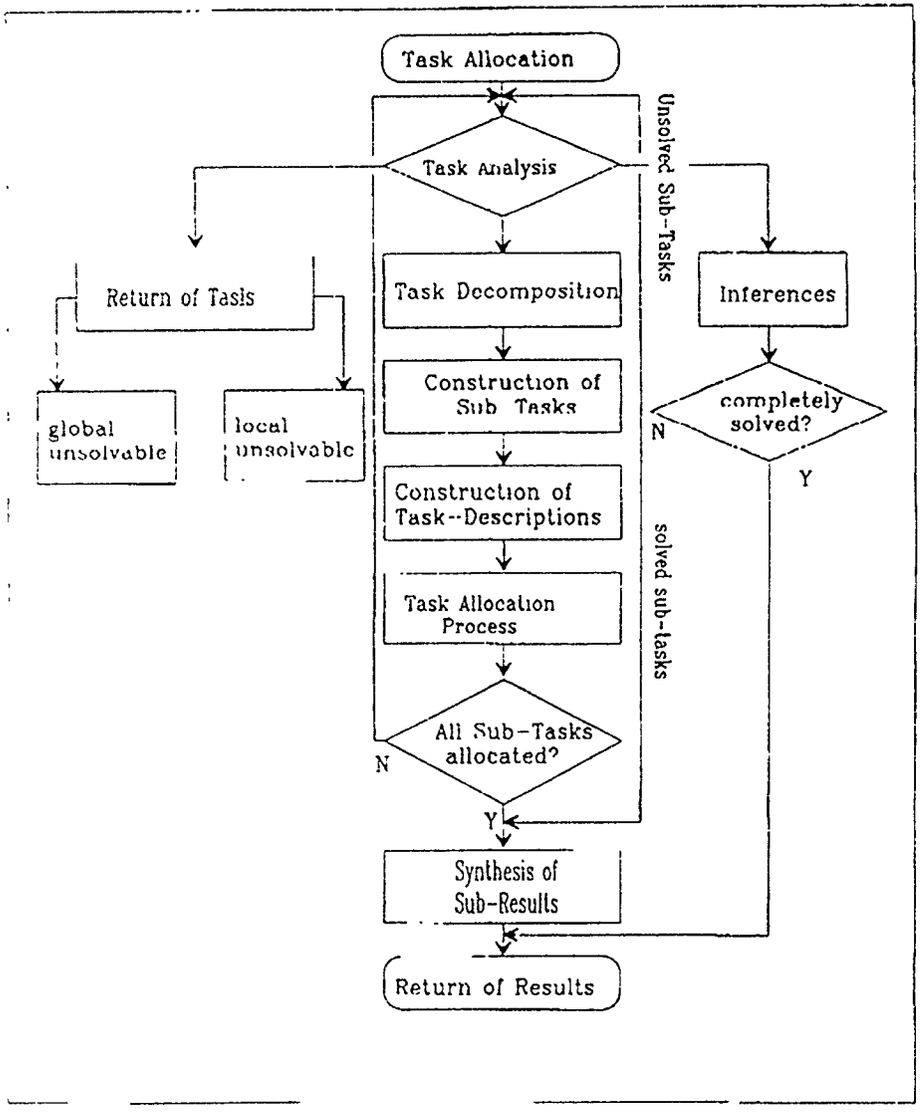


Fig. 3.-1. Local activities during federative problem solving

results, which further serve as basis for task allocation decisions. That means, in respect to task allocation we see the following central requirements of local agents. They must be able

- to estimate and communicate their own problem solving capabilities,
- to decide, if a described task could be solved locally,
- to decompose a task in a set of sub-tasks and
- to allocate a set of sub-tasks to a group of intelligent agents

The process of task allocation itself is based on bilateral negotiations between intelligent agents. In our model each agent knows, what it knows and what it is able to do. Therefore, the process of negotiation is based on descriptions of tasks and problem solving capabilities.

Task allocation leads to further local problem solving processes, i.e., local inferences. If local inferences are terminated the system has to check, if the results could be accepted. For example, PROLOG systems try to verify a given statement. PROLOG inferences continue until they find a proof. They terminate, if there are no more rules to fire or if they found an error.

If the situation in federated expert systems occurs, that there are no acceptable or final results, the local system tries to continue problem solving with task decomposition, too. As a first approach, only unsolvable and undividable subtasks are delegated to other agents. I.e., a new cooperation process has to be initiated.

If all subtasks are solved and returned to their managers, they must be synthesized. Like [Wurr 89] points out, synthetic problems are found to be very hard to solve in general. Until today, there exists a lot of open questions in this area.

Finally, cooperating intelligent agents must possess powerful explanation facilities, which can be used to explain not only local problem solving tasks, but also all steps of the overall problem solving process.

4. Central Features of Agents in Federative Expert Systems

Central parts of our work are due to following research areas: centralized and decentralized architectures of the overall system, internode communication between heterogeneous agents, task analysis, task decomposition and synthesis of results, estimation of expert system problem solving capabilities, description (and match) of tasks and problem solving capabilities, dynamic planning of global problem solving activities, task allocation and development of local explanation facilities with global explanation capabilities.

In this chapter, we concentrate on central local tasks during cooperative inter-agent problem solving. First, we develop techniques to estimate problem solving capabilities of intelligent agents, then we give a brief overview over task decomposition, task description and synthesis of results and discuss some possibilities for decentralized, local control of the overall cooperative problem solving process.

An extensive discussion and detailed analysis of these properties and a lot of additional questions are discussed in [Kirn 90]. Central features of the overall system, especially architectures and common internode languages, presents [Klet 89]

4.1. Estimating Expert System Problem Solving Capabilities

A central question in federative problem solving is that intelligent agents have to know, what they know and what they are able to do. Expert system problem solving capabilities depend on two different, but not completely independent observations. They result from expert systems separation of knowledge bases and inference engines.

Therefore, one possibility to describe the competence of agents is based on the agents knowledge base. [KSWu 90A] discussed first steps towards abstract descriptions of the contents of knowledge bases. As starting point they take those requirements of an agent, which, at least, it has to accomplish to solve a given task. Some of these are concepts and instances of knowledge bases and the relations between them. Others handle with the distribution of concepts and instances over the overall system.
Another, rather obvious idea is the description of expert systems resources like main memory, graphical I/O-devices, parallel processing possibilities, real-time-features and so on.
To describe and to process this knowledge over knowledge [KSWu 90A] defined some criteria and predicates:

- predicate "has resources",
- predicate "has concepts",
- predicate "has instances",
- predicate "same concept",
- predicate "is subset of" and
- predicate "known-all".

The first three predicates define, that those agents aren't be able to solve a given task, which, in respect to this task, don't match these predicates. The predicates four and five characterize simple relations between two concepts. Predicate six says, that an agent possesses all knowledge to a given part of a domain. Examples and further details are discussed in [KSWu 90A].

A second possibility to describe problem solving possibilities of intelligent agents is based on their inference engines. The idea is, that different types of problems may require different types of problem solving strategies. Three central problems are given:

- How could problems be classified (for example, see [Stef 82], [Clan 85], [Pupp 88])?
- How could problem solving strategies be classified?
- How could we relate problems and problem solving strategies?

The last two questions are discussed in [KSWu 90A] and [Kirn 90]. In this area, the point is, what derivation strategies (for example forward and backward chaining, pattern matching) problem solving strategies are given in an expert system (skeleton planing, different searching strategies) and what are the problems (construction, simulation, diagnosis), which require just these strategies?

Today, only simple relations can be defined. For simple cases, we can show, that there exist operational relations between types of problems and problem solving strategies (details are discussed in[Kirn 90]). Some examples are

- predicate "requirements derivation strategy",
- predicate "dynamic variables" (in case of forward chaining expert systems),
- predicate "requirements problem solving strategy",
- predicate "result garanteed".

These predicates are first steps to describe dynamic properties of expert systems. Further results still require a lot of work to be done in the area of classification tasks, a discussion, which, for example, in Germany begins to grow.

A lot of problems occur with the acquisition of this kind of knowledge. The knowledge to define some of the above discussed predicates can be derived automatically (concepts, instances, partially resources), but the knowledge acquisition to define the other predicates must be done by humans themselves. That's why we propose no procedural, but declarative concepts to describe and maintain this knowledge.

With these comments we finish the discussion, how to estimate intelligent agents problem solving capabilities. A detailed overview of related problems, further details and possible approaches to further develop this interesting area of research is given in [Kirn 90].

4.2. Task Decomposition, Task Description and Synthesis of Results

Decomposition of Tasks

Tasks are defined using some language L(G), which itself is defined on basis of a grammar G. If we use grammars, which could be analysed automatically (for example context free grammars, see the above example) task decomposition will use syntax-graphs and could be done by local task decomposition moduls.

Syntax-based task decomposition can lead to situations, which require further analysis. One example are identical sub-graphs, or sub-graphs, which are inverse to one another. The problems we address here are, in some aspects, well known from query optimization in databases.

Task Description

In federative expert systems, task descriptions have to be derived automatically. However, the development of general mechanisms for automatic deduction of abstract task descriptions still yields a wide field of difficult open questions. In a pragmatic view, however, there are some possibilities to reach first results. These possibilities depend directly on the language used to formulate tasks. From the discussions in chapter two, we could derive following examples of task descriptions

- f(x, y). function, which operates on $\mathbb{R} \times \mathbb{R}$,
- f(x, y): function, which has to solve terms like INT E / ABL E / FAK E / ABS E / ... or
- f(x, y): function, which gives results in $\mathbb{R}$.

Synthesis of Results

Like [Wurr 89] points out, the development of automatic mechanisms to solve synthetic problems generally yields severe problems, too. These problems widely depend on the language used to define tasks and results. Using context free or functional languages will help to solve these questions, while object-oriented or frame-based languages tend to complicate synthesize problems.

In our implementation described in chapter five we use a functional language (Natural Expert Language) to represent knowledge, tasks and results, which helps to handle problems of term decomposition and synthesis.

4.3. Task Allocation

Along with [Klet 89], efficient task allocation requires solutions for two different and independent problems. First, the local system has to find out, which set of subproblems should be allocated to other systems. This question deals with the selection of the most appropriate task decomposition, it is solved by a process we name "dynamic planning of global problem solving". Second, the agent should distribute the set of selected subtasks efficiently. That means, for each subtask it should find the most appropriate problem solver.

Dynamic Planning Of Global Problem Solving

Dynamic planning of global problem solving helps to choose the most appropriate set of subtasks from a given set of equivalent sets of subtasks. In this sense, following sets of subtasks from the above example are equivalent:

Overall task

<==> {[INT(1/(cos^2(P(x)/Q(x))-a^2)^{0,5})] + [INT(1/(a^2-x^2)^{0,5})], [ABL(arcsin(Q(y)/a))]}

<==> {[INT(1/(cos^2(P(x)/Q(x))-a^2)^{0,5})], [INT(1/(a^2-x^2)^{0,5})], [ABL[arcsin[Q(y)/a]]]}

<==> {{[INT(1/(cos^2(P(x)/Q(x))-a^2)^{0,5})], [INT(1/(a^2-x^2)^{0,5})], [ABL[arcsin[Q(y)/a]]]}

As we see, the question, which of all possible sets of subtasks is the most appropriate to continue problem solving requires task decompositions with respect to the overall distribution of problem solving capabilities. In this view, syntax-based task decompositions include one important deficiency. They don't take into account the global overall distribution of problem solving capabilities.

In most cases the general problem of finding out all possible sets of subtasks will be found NP-hard. Therefore, we propose a heuristic mechanism to solve this question. In a first step, we decompose a given task and receive its direct subtasks. In a second step for these subtasks we try to find a distribution to appropriate agents. For those subtasks which couldn't be distributed we continue with step one. This algorithm proceeds, until all subtasks can be distributed to other agents or some subtasks can't be decomposed further. If there isn't another syntax graph to try a new way of solving the task distribution problem, the overall task can't be solved completely.

The Task Distribution Problem

Now we assume that the system gets a set of subtasks, which all could be distributed to some agents. Central function of the task distribution modul is the optimal assignment of subtasks to agents, using bilateral negotiation processes. Three different situations may occur ([SMCN 86])

- *There isn't some possibility to assign all tasks (for example, one agent should solve all tasks, but then it couldn't meet given time constraints)*
- *There is just one possibility to assign tasks to agents.*
- *There are more than one possibility to assign tasks to agents*

Only the third situation defines a real optimization problem during task distribution. To solve this problem we need heuristic mechanisms, which could result in inappropriate distribution decisions. This leads to a dynamic component in task distribution, too.

[KSWu 90A] developped an algorithm to solve task distribution problems, which is based on the predicates discussed above, i.e., the algorithm uses a resource-criteria, a concepts-criteria, an instances-criteria and a criteria of dynamic variables. [Klet 89] proposes another approach, based on

descriptors for tasks and knowledge based systems. In this approach there are combined similarity calculations with techniques from decision theory. A third concept could be found using frames to model tasks and agents. Pattern matching then would be an appropriate concept to find out "good pairs" of tasks and agents.

At this point we end our considerations around the area of task allocation. Further discussions, and, from different views, a broad analysis of open questions are published in [KSWu 90A], [KSWu 90B].

5. Federative Cooperation of Expert Systems in Banking: An Implementation Overview

Projecting a federative expert system from existing agents is an interesting idea to couple such systems which can't or shouldn't be integrated in one big overall stand-alone system and

- if there are suitable stand-alone systems,
- if the domain knowledge of these agents could be combind and
- if there are tasks, which are too complex to be solved only by one agent alone, but which could be solved by interacting agents possessing different problem solving capabilities.

Meanwhile, the concepts developed making agents capable to cooperate in a rather flexible, federative manner were implemented in a banking application. In our view, banking possesses some interesting features. There are already a lot of expert systems in use (see, for example, [Ziel 89]). These systems solve tasks from different domains, which, in general are dependent upon one another and their domains often possess identical subsets of knowledge, too. Therefore, we expect interesting advantages from integrating such systems in a federative expert system as described above.

Fig. 5.-1 shows the kernel federative expert system, implemented using NATURAL EXPERT and NATURAL, ADABAS. The five agents are able to solve tasks in the following areas:

- *analytical studies of balance-sheet,*
- *borrowing power of private persons and credit decisions,*
- *capital investment,*
- *analysis of little and mean companies and*
- *analysis of investment structures.*

The agents are completely autonomous. Partially, they possess common subsets of knowledge. Each agent interacts with specialized databases (companies, interest rates, rates of exchange, clients and so on). They try to solve tasks stand-alone. In the actual implementation, a local agent initializes cooperation processes, if it is not able to solve a given task alone. We think, additional motivations to start inter-agent cooperations are needed and possible. Therefore, a part of the actual work is dedicated to answer such questions.

The kernel system decribed in fig. 5.-1 already possesses a great variety of cooperation possibilities. Within an example, we illustrate some potential processes of cooperative problem solving.

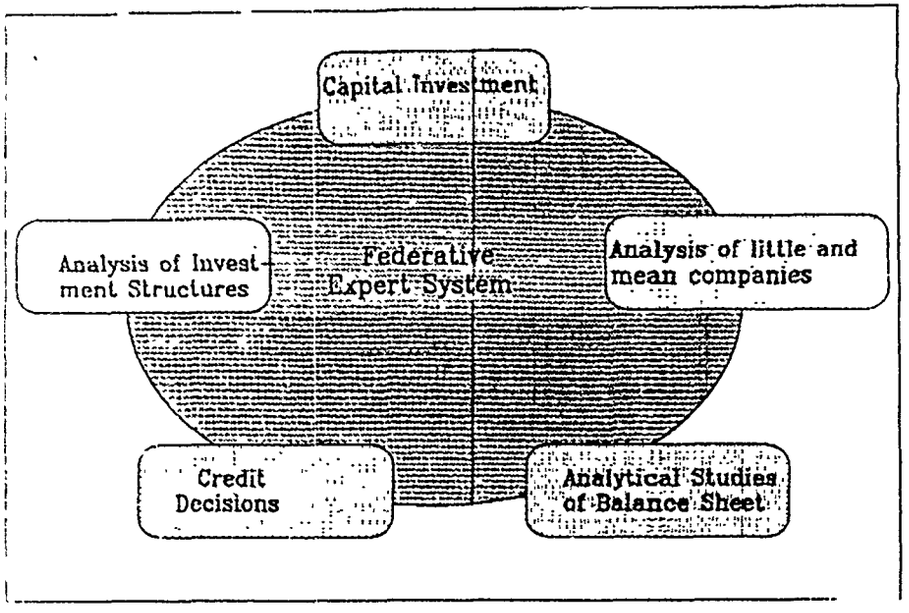


Fig 5.-1: Federative Expert System in Banking

Now take a look at the expert system, which analyses little and mean companies. Such companies could be organizes as capital company or as company with unlimited liability. If we are interested in a companies substance analysis, we must take a look at the legal form of that company. If it is organized as company with unlimited liability, we have to analyse the property of its owners, too. But also, we have to take a look at other important things. For example, the company possesses some other companies. It could possess own properties, which, for example, need a complete investment reorganization and so on In effect, that means, it would be a strong restriction to fix the possibilities of searching agents, which could help to solve a local problem. Instead of this, the agents of the federative expert system analyse tasks dynamically to find out those agents with which they should cooperate to solve the problem in an acceptable way. We see, the same arguments, which led us to the development of expert systems now turn the design of cooperative knowledge processing from (hierarchical) distributed to more flexible, i.e. federative, concepts of cooperative problem solving.

Finally, we point out those technical properties agents must possess, if they should be integrated in a federative expert system. We concentrate the discussion on essential results, details will be published in [Kirn 90]. Besides the requirements to decompose tasks and synthesize results we see the following important aspects.

1. Requirements with respect to inference engines:

- inference processes should allow full parallel knowledge processing (different tasks from different users),
- inference processes should be re-entrant in the sense, that a temporal termination with following inference continuation starting from the termination point should be possible
- there should exist possibilities to control inferences from extern processes.

2. Requirements with respect to the administration of knowledge bases:

- expert system's access to more than one knowledge base,
- multi-user-access to knowledge bases,
- administration of access-rights to knowledge bases,
- transaction concepts, concurrency control mechanisms and recovery componentes
- insertion of new knowledge into knowledge bases during run-time.

Most existing expert system tools possess some of those characteristics which address the inference engine. Product studies however have shown, that the requirements to administrate knowledge bases of cooperating agents are more critical.

6. Conclusions and Further Research

To solve essential problems of expert systems in large and heterogeneous domains this paper proposes a concept of federative expert systems. They are built up from autonomous intelligent agents, which in general solve problems alone, but, if a given task requires cooperative problem solving, they interact in a rather flexible way. The above considerations showed the need for :

- strong concepts to estimate problem solving capabilities of intelligent agents,
- development of techniques to administrate full multi-user-access to knowledge bases like transaction concepts etc. in the field of databases,
- the existence of ful re-entrant inference engines.

The paper discussed first steps towards productive concepts for cooperative problem solving. First results work are given in chapter 3 and 4. Further work now addresses the following questions:

- classifying problems in problem classes and expert systems in problem solving classes, to improve the possibilities to estimate expert system problem solving capabilities.
- development of additional and more flexible cooperation concepts.
- to improve technical characteristics of knowledge base administration.

In addition, we shall integrate new expert systems of heterogeneous technology into the overall system to address questions of common internode languages and problems of translating knowledge from one representation technology to another.

7. References

[AlOt 83] Albert, J., Ottmann, Th.: Automata, Languages and Machines for End-User, Mannheim, Wien Zurich 1983 (in german)

[Bond 88] Bond, A.; Gasser, L. Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[Clan 85] Clancey, W.: Heuristic Classification, AI-Journal 27, 1985, pp. 289.

[Crem 88] Cremers, A.B.; Geisselhardt, W.: Expert Systems Proceedings, Symposium, Univ. of Duisburg, Federal Republic of Germany, 18-19.2.1988, Duisburg 1988 (in german).

[Deen 88] Deen, S.M.: Research Issues in Federated Knowledge Based Systems, British Computer Society Workshop Series: Research and Development in XPS V, Proceedings of XPS, Cambridge 1988.

[FiGa 87] Findler, N.V.; Gao, J.: Dynamic Hierarchical Control for Distributed Problem Solving, Data & Knowledge Engineering 2(1987) pp. 285-301.

[Kirn 90] Kirn, St.: Intelligent Agents and Their Integration in Federated Expert Systems, FernUniversität Hagen, thesis, to appear, Hagen 1990 (in german).

[Klet 89] Klett, G.: Cooperating Expert Systems with Contract Net-Architecture and Their Use in Chemical Applications, FernUniversität Hagen, thesis, Hagen 1989 (in german).

[KSWu 90A] Kirn, St., Schlageter, G., Wu, X.: Meta-Knowledge over the Competence of Federative Expert Systems, FernUniversität Hagen, Computer Science Reports No. 89, 1/1990, Hagen 1989 (in german).

[KSWu 90B] Kirn, St., Schlageter, G.; Wu, X.: Expert Systems in Federative Systems: Problem Analysis, FernUniversität Hagen, Computer Science Reports No.90, 2/1990 (in german)

[Pupp 88] Puppe, F.: Introduction in Expert Systems, Berlin et.al., 1988 (in german)

[ScKl 88] Schlageter, G.; Klett, G.: Federative Knowledge Based Systems, in [Crem 88], pp. 204.

[SMCN 86] Steeb, R.; McArthur, D.J.; Cammarata, S., Narain, S., Giarla, W.D.: Distributed Problem Solving for Air Fleet Control: Framework and Implementation, in [Klah 86], pp. 391.

[SmDa 78] Smith, R.G.; Davis, R.: Application of a Contract Net, Framework: distributes Sensing, Proc. Arpa Distributed Sensing Net Symposium, Pittsburgh 1978, pp. 12-20.

[Smit 80] Smith, R.G.: The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980

[Softw 90] Software AG: Natural Expert Users Guide / Reference Manual, Darmstadt 1990

[Stef 82] Stefik, M. et. al.: The Organization of Expert Systems - A Tutorial, AI Journal 18, 1982, pp. 135, auch in Hayes-Roth, F., Waterman, D., Lenat, D. (eds.): Building Expert Systems, Addison-Wesley, 1983.

[Wied 89] Wiederhold, G. et al.: Partitioning and Composing of Knowledge, Stanford University, 1989

[Wurr 89] Wurr, P.: Model for Synthetic Tasks, Computer Magazin, 6-7/1989, pp. 51 (in german).

[Ziel 89] Zielenziger, D.: Econometric Modeling For Traders on a Budget, Global Finance, pp. 70, 1989.

A COOPERATION FRAMEWORK FOR INDUSTRIAL PROCESS CONTROL

C. Roda, N. Jennings, E.H. Mamdani (1)
QMW University of London, Dept. Elect. Engineering
Mile End Road, London E1 4NS

1 INTRODUCTION

Examination of the Distributed AI literature [Bon88, Gas89, Huh88] reveals that a significant proportion of existing cooperative systems are purpose-built to solve particular problems using techniques which are appropriate for that domain (e.g. Air Traffic Control [Cam83], Vehicle Monitoring [Dur88] and speech recognition [Erm75]). However the ARCHON[2] project aims to be less related to one specific domain, providing a cooperation framework suitable for industrial process control applications in general.

There are several characteristics of this domain which impact upon the design of the framework. Firstly, there is a substantial amount of existing domain-level[3] software, therefore pre-existing systems as well as problem solvers specifically built with this cooperation framework must be capable of being incorporated. The construction of a multi-agent community can either be seen as a top-down process aiming to create "a team of cooperating agents that act together to solve a single task" [Lec87] or as a bottom-up process aiming to construct an environment in which agents who justify their own existence[4] [Mul89] can cooperate. A bottom-up approach was adopted because of the need to incorporate existing software and also because the chosen applications do not solve one single task which can be easily decomposed.

In order for agents to justify their own existence they need to be capable of a significant amount of computation (coarse granularity [Sri87]) - this complexity justifies the sophistication of the cooperation framework. It would be inefficient to construct such a framework for simple problem solvers because an overly large proportion of its activity would be spent in coordination rather than actual problem solving. The second domain characteristic is that of heterogeneity, most work presently documented deals exclusively with homogeneous agents, however in industrial process control this is not a realistic assumption. Typical domain level systems occurring in process control include: expert systems, planners, databases and conventional numerical software. This means the framework must provide appropriate facilities for interaction between all of these system types.

Design goals (increased reliability and enhanced problem solving capabilities in this case) have a major influence on system structure. These aims lead to two forms of distribution - namely physical distribution and decentralisation. Reliability criteria imply the need for physical distribution of agents; having the community distributed over several machines means that failure of a single component is less significant.

Reliability also implies that the system should have decentralised problem solving - hence the community is organised as a "group" [Fox81] of problem solvers. Characteristics of groups include a limited number of members (few agents in the community) and the absence of an explicit control hierarchy. Control hierarchies are less reliable because they do not exhibit graceful degradation of performance; failures at level n of the hierarchy isolate subordinates at level n+1 and below.

To enhance problem solving whilst maintaining reliability and fighting complexity, it was decided to structure the system as a community of cooperating agents. Decentralization and the group organisational strategy implies that individual agents have a limited view of the overall problem (bounded rationality [Sim57]) and that coordinated behaviour is hard to guarantee [Dav83]. Therefore mechanisms must exist within the framework for coordinating community activity, such mechanisms should ensure that:

- Misleading and distracting hypotheses are not spread
- Multiple agents do not compete for or try to access resources simultaneously
- Agents do not undo the results of another
- Actions are not carried out redundantly

In the remainder of the paper communication as a support mechanism for coordination and three aspects of coordination itself (task decomposition, urgency and uncertainty) are analysed. The structures described assume that each agent has some knowledge of other "interesting" community members (acquaintances) - see section 2.

2 ACQUAINTANCE MODELLING

It is now a well established fact (in DAI, psychology, sociology and other disciplines which examine cooperative, group problem solving) that in order for sophisticated cooperation to occur, agents need to be able to reflect about their role and also that of others within the community. One important aspect which impacts upon this reflective process is the knowledge maintained

(1) This paper reflects the effort of the whole ARCHON consortium whose partners are: Krupp Atlas Elektronik, JRC Ispra, Framentec, QMW, IRIDIA, Iberduero, ERDC, Amber, Technical University of Athens, University of Amsterdam, Volmac, CERN and University of Porto.

2 ARCHON stands for "Architecture for Cooperative, Heterogeneous On-line systems" and is an ongoing ESPRIT II project P2256.

3 E.g. in electricity management applications: high/low voltage diagnosis expert systems and databases containing network topology.

4 Contrast this with the elementary problem solvers present in the neural network paradigm [McC86] or blackboard knowledge sources [Hay83, Erm75] which are rendered useless when removed from the community.

about other community members. As yet no consensus has emerged; however by examining existing systems several strands of commonality appear [adapted from Bra89 and Wer89]:

- State information provides an indication about the approximate state of processing of other community members.
- Role knowledge represents stable information about other community members (eg capabilities, strategies and areas of expertise).
- Intentional knowledge indicates what an agent intends to do - usually described in terms of plans.
- Evaluative knowledge is a form of meta-knowledge which permits agents to distinguish between agents which offer similar services. It may include a competence rating for skills, a measure of reliability attached to information coming from a particular agent or a measure of punctuality in a time-critical environment.

It is envisaged that the above types of information will also be embodied in the ARCHON acquaintance models, which each agent will maintain. However this is still very much an ad hoc approach and ongoing work within the project aims to formalise the types of knowledge required to support different modes of cooperative interaction (eg task-sharing, result sharing [Smi81] and coordinating behaviour).

3.THE COMMUNICATION PLATFORM

The aim of the ARCHON Communication Platform is to support distribution at both a physical and a conceptual level. At the physical level, distribution is supported by providing mechanisms for information exchange amongst agents resident on different machines. At the conceptual level, distribution is supported by a set of functions which allow acquaintances to establish meaningful dialogues in order to allow decentralized problem solving.

A physical communication service is necessary for any community composed of agents resident on different physical media. However this is not the case for conceptual distribution support; since communication strategies are normally incorporated in the control mechanism of each agent, conceptual communication platforms are not always explicit. ARCHON allows the multi-agent system builder to use conceptual communication functions (e.g. "send this message to all agents which are interested in it") as well as physical communication functions (e.g. classical send and receive). In ARCHON both platforms are based on a message exchange paradigm; shared memory systems were considered less robust for the target environment.

The physical communication platform is based on the OSI standard. The session layer service as defined in the CCITT recommendation X.200 has been extended to support the types of communication required for cooperation. It assumes the existence of a communication mediator which acts as a "Conference Co-ordinator". This structure allows point-to-point, point-to-multipoint and broadcast communication as well as providing storage and retrieval of messages and transparent synchronization.

The conceptual communication platform adds one level of abstraction to the physical communication platform, providing: intelligent addressing, filtering, scheduling.

Intelligent addressing allows agents to send messages to "relevant" acquaintances. For example if information has been generated by agent A1, it may want to send it to all agents which are interested in it. Intelligent addressing mechanisms are based on parameters which specify how to decide the relevance of a message to an acquaintance, and on a description of its knowledge and goals (i.e. information embodied in the acquaintance's models).

Filtering means that agents only receive relevant messages - "intelligent addressing" can be seen as filtering out-going messages. An agent can use filtering to receive messages only from certain acquaintances or about certain subjects, for instance.

Scheduling mechanisms in the conceptual communication platform, allow agents to receive messages in a specified order. The order being defined on the basis of message type (e.g. messages carrying an answer to a previous question, messages carrying information spontaneously supplied by other agents or messages carrying requests from other agents) or on the basis of an abstract description of the message's content. The need for message abstracts representing domain knowledge occurs in the three types of conceptual communication described. Both filtering (in/out) and scheduling may need such domain knowledge. An example of the use of this knowledge is the Time-Out service which enables communication to be aborted when a message becomes obsolete. In general the evaluation of the timeliness [Dur88] of a message is strictly related to its meaning in the domain of the application.

4 COOPERATION FRAMEWORK

This section describes three aspects of cooperation: coordinating activity, urgency and uncertainty. We believe that these issues need to be tackled by any multi-agent system and therefore it is important to find domain independent functions which support their management.

4.1. COORDINATING ACTIVITY

When designing multi-agent systems in a top-down manner, problem decomposition and allocation of tasks to appropriate agents is one of the primary considerations[5]. However because of the bottom-up design philosophy adopted, problem decomposition is pre-determined by the domain level problem solvers present in the community. For example, if goal G can only be achieved by a single agent then task allocation is trivial. However if multiple agents can achieve G then task allocation becomes more complex and necessitates reasoning about acquaintances[6].

With the chosen design philosophy coordinating activity within the community becomes the major issue. If agents merely pursue their own goals then performance of the community as a whole is likely to degenerate. Consider the situation of distributed traffic light control where each agent controls a single traffic junction. The overall aim of the system may be to keep traffic flowing on the main roads and hence minor roads feeding into the main road may have to work non-optimally in order to achieve the desired system-wide performance characteristics. This scenario is an illustration of the cooperation layer acting as a distributed reasoning system and reasoning about the community as a whole. In this case, coordinating behaviour achieves the goal which no individual entity is capable of - an alternative way of achieving this is to centralise and have a dedicated agent.

A second type of coordination activity involves agents reasoning about the behaviour and activities of two or more acquaintances (rather than of coordination within the whole community). Such behaviour aims to ensure that agents do not undo work of others (eg one robot continually moving a block onto the floor and another continuously placing it back in its original position)

Therefore agents require mechanisms and information for reasoning about the goals, skills and plans of other community members. The representation and management of global and local optima is part of the future work of the project.

4.2. URGENCY

In the course of our research we observed that a clarification of the "coupling level" concept was necessary to evaluate urgency of agents' activities. Many references are made to it but all the given definitions are limited to intuitive descriptions. The coupling level concept is also related to the autonomy concept.

Studying coupling and autonomy, we have come to the conclusion that they respectively give a quantitative and a qualitative description of the agents' activity. We say that an agent is tightly coupled to another (set of) agent(s) if it spends the majority of its time waiting for information / resources / activities which other agents control. Therefore the level of coupling measures the percentages of time spent by an agent waiting for some event controlled by another agent to happen. Autonomy is defined by looking at the activities an agent can accomplish between waiting times. If this activity can be seen as "complete" (e.g. it solves a subgoal), then the agent is autonomous. Whilst the coupling level measures the "quantity" of activity an agent is able to perform independently from its acquaintances, the autonomy level describes the "quality" of this activity. We will concentrate on the former measure which can be specified with a domain independent number, i.e.. agents may be called loosely coupled if less then a predefined percentage of their time is spent waiting. The autonomy measure is strictly related to the domain of the application the agents are working at.

The treatment of urgency (task and message scheduling) is seen as a means by which agents can be kept loosely coupled. Dependencies between agents' activities must be recognized and dynamically reduced by a selective choice between different forms of eager or lazy evaluation. It is recognised that the coupling level in multi-agent systems is strongly related to the design of the domain level systems (and since we are adopting a bottom-up approach this affects us). For instance, loose coupling is often not achievable when each agent has different capabilities and all the capabilities are necessary to solve the problem, see the example described by figures 2.1 and 2.2. However, assuming a domain level design which permits decoupling to some extent, dynamic management of dependencies can improve the evaluation of urgency in task/message scheduling.

Let us see two examples that should clarify the aforementioned concepts.

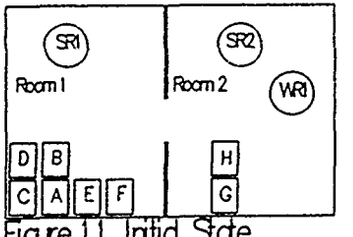


Figure 1.1 Initial State

5 Much of this work is termed "Distributed Problem Solving" and considers how the work of solving a particular problem can be divided among a number of modules that cooperate [Gas89].

6 If multiple agents can achieve goal G and they all use the same methods and resources, then selection of the agent to achieve G can only be based on state information (eg workload, current activities). However if the achievement of G differs in some respect, then knowledge about the differences can also be used to make the decision (evaluative knowledge).

Figure 1.1 describes the initial state in a domain in which three robots operate. Still Robots SR1 and SR2 are able to *stack* and *unstack* blocks which are in the same room. Walking Robot WR1 is able to *take* a block from the floor of one room to the floor of another one. Whilst *stack* and *unstack* operators are atomic, *take* is achieved via the execution of three actions, namely *pick-up*, *goto*, *put-down*. Figure 1.2 describes the final state the agents want to achieve (goal-state); figure 1.3 gives a STRIP-like [Fik71, Fik71a, Fik72] description of the operators used.

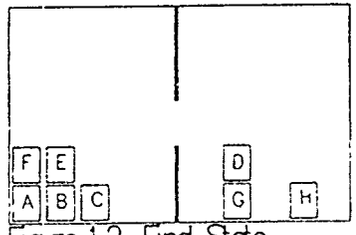


Figure 1.2 Find State

The following graphs 1.1, 1.2, 1.3 show three possible activities sequences for the agents. All three sequences are solutions in that they allow a transaction from the initial state to the final state. Three parameters are used to measure the quality of each solution:

1. TIME necessary to complete the goal
2. TOTAL EFFORT: total number of atomic actions executed by the agents
3. COUPLING LEVEL: percentage of time in which each agent has been active between the beginning and the end of its activity.

An improvement in execution time will be possible if either the total effort is augmented or if the agents spend less time waiting (they are more loosely coupled). Table 1 gives the value of time, total effort and coupling level for each one of the three solutions. It is shown here, that appropriate scheduling mechanisms may improve both the execution time and the coupling level whilst leaving the total effort unchanged.

In the solution described in graph 1.2, SR1 starts by unstacking block D enabling WR1 to take it in room2 and, as a consequence, SR2 has a lower waiting time. To choose the correct action SR1 must know that the execution of UNSTACK(SR1,D,C,ROOM1) leads to ON-FLOOR(D) which is the precondition for WR1 to execute PICK-UP(D). If SR1 doesn't have this knowledge then the situation of graph 1.1 may occur. If SR1 sees that WR1 will want to PICK-UP D then the optimum may be achieved as described in graph 1.2. Graph 1.3 shows the situation in which SR1 is only able to recognise that WR1 requires D ON-FLOOR when WR1 explicitly starts waiting for the state ON-FLOOR(D).

STACK(ROBOT BLKx BLKy ROOM)				
PRE	INROOM ROBOT ROOM		CLEAR BLKx	ON FLOOR BLKY
	INROOM BLKx ROOM		CLEAR BLKy	
	INROOM BLKy ROOM		ON-FLOOR BLKx	
ADD POST	ON BLKx BLKy ROOM			
DEL POST	CLEAR BLKy	ON FLOOR BLKx		
PICK-UP(ROBOT BLKx ROOM)				
PRE	INROOM ROBOT ROOM		CLEAR BLKx	
	INROOM BLKx ROOM		ON FLOOR BLKx	
ADD-POST	HOLDING ROBOT BLKx			
DEL POST	ON FLOOR BLKx			
UNSTACK(ROBOT,BLKx BLKy ROOM)				
PRE	INROOM ROBOT ROOM		INROOM BLKx ROOM	
	ON BLKx BLKy			
ADD POST	CLEAR BLKy	ON-FLOOR BLKx		
DEL POST	ON BLKx BLKy			
GOTO(ROBOT ROOMx ROOMy)				
PRE	INROOM ROBOT ROOMx			
ADD POST	INROOM ROBOT ROOMy			
DEL-POST	INROOM ROBOT ROOMx			
PUT-DOWN(ROBOT BLK ROOM)				
PRE	INROOM ROBOT ROOM		HOLDING ROBOT BLK	
ADD-POST	ON FLOOR BLK			
DEL-POST	HOLDING ROBOT BLK			

Figure 1.3 Operators

Further options may also be considered, e g. WR1 may itself explicitly require that SR1 UNSTACK(D). The domain of example 1 is a typical bottom-up domain. SR1, SR2 and WR1 are given agents which justify their own existence and have fixed capabilities. Task allocation is trivial (each sub-goal can only be solved by a single agent) and coordination is the major issue. Therefore each agent, solving its own sub-goals, must reason on the consequences that its actions have on other agents' activities. In particular this section has described how appropriate scheduling of action may improve the overall result.

SOLUTION 1		
Total Time = 8	Total effort = 9	
SR1 Time = 4	SR1% = 100%	
SR2 Time = 8	SR2% = 25%	
WR1 Time = 7	WR1% = 43%	
SOLUTION 2		
Total Time = 5	Total effort = 9	
SR1 Time = 4	SR1% = 100%	
SR2 Time = 5	SR2% = 40%	
WR1 Time = 4	WR1% = 75%	
SOLUTION 3		
Total Time = 6	Total effort = 9	
SR1 Time = 4	SR1% = 100%	
SR2 Time = 6	SR2% = 33%	
WR1 Time = 5	WR1% = 60%	

Table 1 Solutions Evaluation

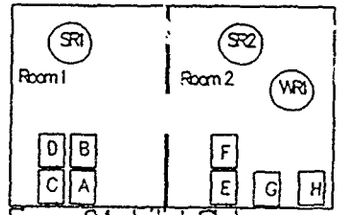


Figure 2.1 Intid State

A second example described in Figures 2.1 and 2.2, shows a case in which given the domain and the particular problem it isn't possible to improve the coupling level that agents display in the obvious solution.

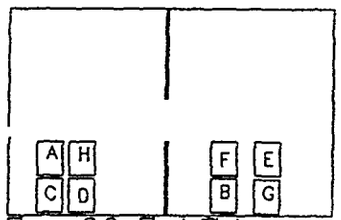


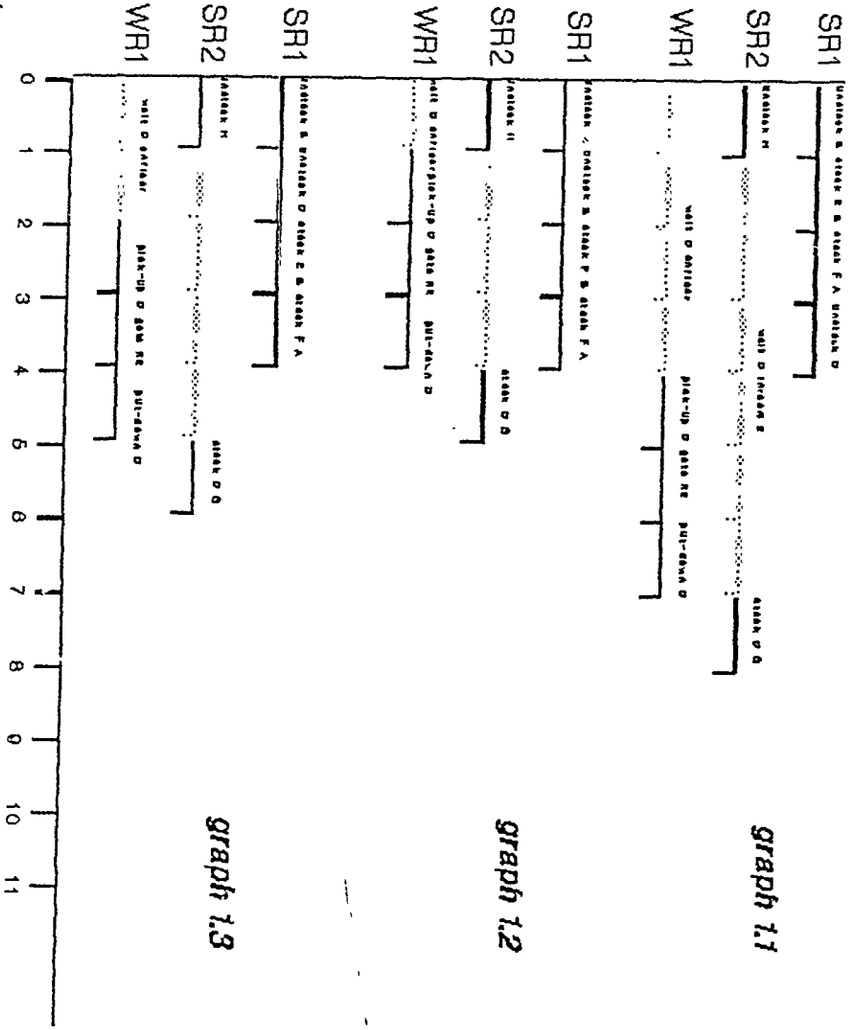
Figure 2.2 Find State

3.3. *UNCERTAINTY*

Uncertainty in a multi-agent community can be an inherent property of the domain, implying agents must be able to exchange uncertain information about the problem domain and manage it even in presence of heterogeneity. Another source of uncertainty is caused by the interaction amongst agents (*organizational uncertainty*).

The first uncertainty problem is related to heterogeneous representations of *ignorance / fuzziness* at the domain level. Since we are assuming that the domain level problem solvers may have been designed as systems in their own right, they may have completely different *semantics*. Therefore the heterogeneoty may be due to different "points of view" or to different methodologies used to integrate uncertain information in the reasoning system. Consider the sentence:

"It will be cold with probability .75" (1)



graph 1.1 graph 1.2 graph 1.3

Suppose that a Weather Forecast Agent (WFA) passes this information to a Traffic Forecast Agent (TFA). Also assume that both the WFA and the TFA measure the truth of their sentences and inferences using probabilistics methods. Even in this assumption we can find at least two cases in which different "points of view" (semantic meaning attached to syntactic information) may lead to incorrect understanding of the sentence communicated. Firstly it may well be the case that TFA's and WFA's probability measures relate to different scales, e.g. WFA measures in a 0-1 scale and TFA measures on a 0-10 scale). Secondly the temperature values associated with "cold" may differ in the two agents, e.g. WFA may consider "cold" temperatures ranging from -1 to 15 degrees and TFA considers "cold" temperatures ranging from -5 to 7 degrees. If the two agents use different methods to represent and manage uncertain information (e.g. TFA uses Bayesian methods and TFA uses credibility functions) the problem of the correct interpretation of sentence (1) is even more complex. We believe that a possible approach to the problem is to decouple uncertainty treatment and reasoning techniques. This issues has been studied in the consortium and for a discussion of it the reader may refer to [Saf90].

Uncertainty due to interaction may occur even when the application domain is completely well specified and all information is certain. A first type of interaction-related uncertainty is due to communication failures or delays. For instance TFA may require information from WFA, it may receive an acknowledgement saying that WFA will supply the information, even so TFA:

- cannot be certain to receive the information because the communication channel may go out of service;
- cannot be certain to receive the correct information because of transmission errors;
- cannot be certain about the time when it will receive the information because of transmission delays.

In [Hal86] Halpern and Moses show that "not only is common knowledge not attainable in systems where communication is not guaranteed, it is also not attainable in systems where communication is guaranteed, as long as there is some uncertainty in message delivery time".

A second type of interaction-related uncertainty is concerned with acquaintance reliability e.g. TFA may believe that the information supplied by WFA is not completely certain. A third type of uncertainty is due to the fact that each agent cannot be sure about when and if an acquaintance will undertaken a required activity.

Durfee et al. [Dur85] note that it is possible to reduce interaction related uncertainty by appropriate exchange of information. For example information about uncertainty of knowledge can be used to synthesize results or allocate tasks [Dav83, Smi81, Dur88]. In ARCHON, the possibility of attaching "uncertainty tags" to each item of knowledge exchanged is being investigated. This tagging applies to both domain dependent and domain independent knowledge. In the latter case the cooperation framework will generate the tag, whilst in the former the tag should be created by the domain level system.

4. CONCLUSIONS

Some issues related to the construction of a cooperation framework suitable for industrial process control applications have been described. The cooperation framework is based on a bottom-up approach which allows existing systems to be integrated into the community. Whilst subjects such as problem decomposition become less central in this perspective, others such as communication, urgency and uncertainty become more complex to manage and vital to the system. The need for explicit knowledge about other agents has been explained along with the information types that should be part of it. The decoupling of communication into a physical and a conceptual service has been described. Chapter four explains why coordination becomes a major issues in a decentralized multi-agent system composed of pre-existing agents and two aspects of coordination have been stressed: urgency and uncertainty management. Informal definitions of coupling level and autonomy have been postulated with the aim of showing how appropriate scheduling may lead to more loosely coupled systems. Finally problems related to uncertainty management in an environment of heterogeneous agents have been pointed out.

Whilst this paper doesn't aim to give solutions to any of the problems presented, the authors believe that it contains useful insights into what we consider the main issues for a bottom-up approach to multi-agents systems design.

REFERENCES

[Bon88] Bond,A.H. and Gasser,L., (1988), "Readings in Distributed Artificial Intelligence", Morgan Kaufmann.

[Bra89] Brandau,R. and Weihmayer,R., (1989), "Heterogeneous Multiagent Cooperative Problem Solving in a Telecommunication Network Domain", Proc DAI Workshop, pp 41-57.

[Cam83] Cammarata,S., McArthur,D. and Steeb,R., (1983), "Strategies of Cooperation in Distributed Problem Solving", IJCAI 1983, pp 767-770.

[Dav83] Davis,R. and Smith,R.G., (1983), "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence, 20, pp 63-109.

[Dec87] Decker,K.S., (1987), "Distributed Problem Solving Techniques: A Survey", IEEE SMC, 17, 5, Sept 87, pp 729-740.

[Dur88] Durfee,E.H., Lesser,V.R. and Corkill,D.D., (1988), "Cooperation through Communication in a Distributed Problem Solving Network", in [Huh88], pp 29-59.

[Dur85] Durfee,E.H., Lesser,V.R. and Corkill,D.D.,(1985), "Coherent cooperation among communicating problem solvers". Proc. 1985 DAI Workshop, pp. 231 - 276.

[Erm75] Erman L.D., Lesser V.R., (1975) "A Multi-Level Organization for Problem Solving using Many Diverse Cooperating Sources of Knowledge", IJCAI.

[Fik71] Fikes R.E., Nilsson N.J.,(1971) "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". Artificial Intelligence 2, pp.189-208

[Fik71a] Fikes R.E., (1971) "Monitored Execution of Robot Plans Produced by STRIPS". Proc. IFIP 1971, Ljubljana, Yugoslavia (August 1971).

[Fik72] Fikes R.E., Hart P.E., Nilsson N.J. (1972) "Learning and Executing Generalized Robot Plans". Artificial Intelligence, Vol.3, N.4, pp.251-288.

[Fox81] Fox M.S.,(1981) "An Organization View of Distributed Systems", IEEE SMC,11,1.

[Gas89] Gasser,I. and Huhns,M.N., (1989), "Distributed Artificial Intelligence Volume II", Pitman Publishing.

[Hal86] Halpern,J.Y., (1986), "Theoretical Aspects of Reasoning About Knowledge" Proceed. of the 1986 Conference, Morgan Kaufmann Publ., San Mateo, Ca, .

[Hay83] Hayes-Roth B., (1983) "The Blackboard Architecture: a General Framework for Problem Solving ?", Stanford Heuristic Programming Project, HPP-83-30

[Huh88] Huhns,M.N., (1988), "Distributed Artificial Intelligence", Pitman Publishing.

[McC86] McClelland,J.L. and Rumelhart,D.E., (1986), "Parallel Distributed Processing", MIT Press.

[Mul89] Muller,J. and Demazeau,Y., (1989), "Decentralized Artificial Intelligence", Proc. of First European Workshop on Modelling an Autonomous Agent in a Multi-Agent World.

[Sim57] Simon, H.A., (1957) "Models of Man" Wiley

[Smi81] Smith,R.G. and Davis,R., (1981), "Framework for Cooperation in Distributed Problem Solving", IEEE SMC, 11, 1, pp 61-70.

[Sri87] Sridharan,N.S., (1987), "1986 Workshop on Distributed AI", AI Magazine, Fall 1987, pp. 75-85.

[Saf90] Saffiotti,A., (1990), "A Hybrid Framework for Representing Uncertain Knowledge", Proc. of Eight National Conference on Artificial Intelligence AAAI-90, pp 653-658.

[Wer89] Werner,E., (1989), "Cooperating Agents: A Unified Theory and Social Structure", in [Gas89] pp 3-36.

A Working Conference On
COOPERATING KNOWLEDGE BASED SYSTEMS

A Cooperative Architecture for Intelligent Process Control

Ricardo Sanz, Agustín Jiménez and Ramón Galán

Departamento de Automática, Ingeniería Electrónica e
Informática Industrial
UNIVERSIDAD POLITECNICA DE MADRID

Please use the following address for mailing:

Name: Ricardo Sanz
Address: Dept. Automática
ETSI Industriales
José Gutierrez Abascal 2
28006 MADRID
SPAIN

Tel: +34-(9)-1-261 69 89
Fax: +34-(9)-1-564 2961
E-Mail: sanz@disam.upm.es

A Cooperative Architecture for Intelligent Process Control

Ricardo Sanz, Agustín Jiménez and Ramón Galán

Departamento de Automática, Ingeniería Electrónica e
Informática Industrial
UNIVERSIDAD POLITECNICA DE MADRID

Keywords: Intelligent control, cooperative architectures, model based reasoning, artificial intelligence.

EXTENDED ABSTRACT

Introduction

CONEX is an architecture designed for embedded operation in complex process control systems. This is a cooperative architecture with distributed functionality ranging from fuzzy rule-based controllers to high-level model-based reasoning.

This architecture is under development in a contract with a process industry firm. The design objectives for this architecture are:

- Enhancing process operation: production, reliability, availability.
- Reusability: system customization.

The initial installation of the system is a cement kiln control facility.

System Components

The CONEX system is composed by a series of cooperating independent elements. This elements -called HLOs (High Level Objects)- are the following:

- CONEX-PI: Process Interface.
- CONEX-DC: Direct Control.
- CONEX-PM: Process Monitor.
- CONEX-MM: Model Manager.
- CONEX-EC: Expert Controller.

- CONEX-S: Simulator.
- CONEX-UI: User Interface.
- CONEX-SM: System Monitor.

They are implemented using OOP methodologies, so, the communication between them is by means of message passing. The protocols of the HLOs are restricted and use restriction lists.

Operation modes

The system has two fundamental modes of operation:

- Low level operation. Performed by CONEX-PI, CONEX-PM and CONEX-DC.
- High level operation. Performed by CONEX-EC, CONEX-MM and CONEX-S is built on top of low level operation.

This twofold operation has been designed with reliability objectives in mind. The failure of one the high level HLOs imply a graceful degradation of performance of the system, being privated of the top level reasoning mechanisms.

The control action determination activities are performed by means of the cooperation between the HLOs. In normal operation the control actions are derived from a hierarchy of control determination modes. This hierarchy is:

- First, a direct control strategy implemented by means of classic PID controllers or fuzzy rule-based controllers.
- Second, a rule-based expert controller, using an domain specific control rulebase.
- Third, a model based expert controller, using a metacontrol knowledge base and simulation capabilities -qualitative and quantitative- for evaluation of alternatives.
- Fourth, a top level operator, whose intervention is required in case of unrecognizable process situation.

The first one corresponds to low level operation and the following three to high level operation. The use of high level modes is done by request from the process monitor. This object performs continuous pattern recognition to detect abnormal plant situations. This pattern recognition is multiresolutional,

and with temporal focalization.

Direct Control Strategies

At this level the system performs the following tasks.

- Sensor reading and filtering.
- Sensor readings consistence analysis.
- Model-based unsensed variables estimation.
- Control algorithms.
- Actuation.

The control algorithms are implemented in two ways:

- Using classic PID controllers for those loops with no control problems.
- Using fuzzy rule-based controllers for problematic loops.

In case of top level system crash the operation of this strategies continues, but with degradation because the process model isn't available.

Rule-based expert controller

This is a classic rule-based controller. Is implemented in a classic way but with real-time operation characteristics. There exists a control knowledge base which contains operational knowledge extracted from plant operators. This rulebase permits the determination of control actions for specific plant situations.

Model based expert controller

This top level reasoning methodology is used in the system for safety considerations. In real-time controllers expert system cliff effect is unacceptable, so there must exist a methodology for control action determination when process situation falls out of scope of the control rulebase of the former level.

This top level control determination strategy is based on:

- A model based inference engine.
- A multiresolutional model of the plant
- A metacontrol knowledge base.
- A multiresolutional simulator.

The process is the following. The inference engine, using methods of the metacontrol knowledge base and plant information from the model determines one or several control alternatives. The determination of the adequacy of these alternatives and the selection of the best one is made by means of simulation and evaluation of plant evolutions.

Operator

If all the former levels of control fails in the determination of the control action, a operator request is issued. The operator specifies the control action and the system stores it, in order to make situation-action learning.

Cooperation and security

The overall behavior of the system results from the cooperation between the HLOs. Each HLO has an specific activity which may be recalled by several other HLOs.

This approach permits a secure operation, because failure in one of the high level systems -the most failure prone- causes a graceful degradation of system performance, not a global crash

Bibliography

[Alty 87] The Limitations of Rule-Based Expert Systems. J. L. Alty. *Knowledge Based Expert Systems in Industry*. Ellis Horwood, England 1987, pp. 12-16.

[Årzén 89] An Architecture for Expert System Based Feedback Control. Karl-Erik Årzen. *Automática*, Vol. 25, No. 6, pp. 813-827. Pergamon Journals Ltd. 1989.

[Åstrom 86] Expert Control. K.J. Åstrom, J.J. Anton & K.E. Årzen. *Automática*, Vol. 22, pp. 277-286. Pergamon Journals Ltd. 1986.

[Bobrow 84] *Qualitative Reasoning about Physical Systems*. Daniel G. Bobrow (Ed). North-Holland. 1984. Reprinted from Artificial Intelligence, Vol. 24.

[Efstahtiou 87] Rule-Based Process Control Using Fuzzy Logic. J. Efstahtiou. *Approximate Reasoning in Intelligent Systems, Decision and Control*. Elie Sanchez y Lotfi A. Zadeh (Eds). Pergamon Press, pp. 145-158, 1987.

[Ellis 89] Active Objects: Realities and Possibilities. Clarence A. Ellis y Simon J. Gibbs. En *Object Oriented Concepts, Databases and Applications*, Won Kim y Frederick H. Lochowsky (Eds). ACM Press, pp. 561-572, 1989.

[Jakob 90] Situation Assessment for Process Control. François Jakob y Pierre Suslenschi. *IEEE Expert*, April 1990, pp. 49-59.

[King 88] Intelligent Control in the Cement Industry. Robert King. *IFAC Conference on Distributed Intelligence*, Varna, Bulgaria, June 1988.

[Rouse 83] Models of Human Problem Solving: detection, Diagnosis and Compensation for System Failures. William B. Rouse. *Automática*, Vol. 19, No. 6, pp. 613-625, 1983.

[Scarl 87] Diagnosis and Sensor Validation Through Knowledge of Structure and Function. E.A. Scarl, J.R. Jamieson y C.I. Delaune. *IEEE Systems, Man and Cybernetics*, Vol. 17, No. 3, pp. 360-368. 1987.

[Shirley 87] Some Lessons Learned Using Expert Systems for Process Control. Richard T. Shirley. *IEEE Control Systems Magazine*. December 1987.

[Simmonds 88] Representation of Real Knowledge for Real-Time Use. W.H. Simmonds. *IFAC Workshop on AI in Real Time Control*. 1988.

[Voss 88] Architectural Issues for Expert Systems in Real Time Control. H. Voss. *IFAC Workshop on AI in Real Time Control*. 1988.

KNOWLEDGE BASED ELECTRONIC MAIL SYSTEM

M.Kantardzic, N.Milicic, A Filipovic, V.Okanovic
Laboratory for electronics and computer science
Faculty of Electrical Engineering Sarajevo
Toplicka bb, 71000 SARAJEVO, YUGOSLAVIA

ABSTRACT

In this paper we expose the use of a *knowledge base of documents and users* in electronic mail system, offering thus a wide set of useful and more automated operations, in very comfortable way. In fact, all the advanced mail operations provided by our system, are based on good and precise description of documents and users. Such operations enable people to easily find useful information about existing documents, to send the documents without specifying user's address and to prioritize, sort, select, discharge and forward his mail on the automatic way.

1. INTRODUCTION

Analysis of the research papers [7,8], as well as practical implementation concerning the area of electronic mail, has shown that most of them have emphasized standard mail function. Such examples are sending, receiving [2] and additional procedure related to processing already present mail. The most common additional procedures are sorting and giving priority to mail, automatically answering to certain messages [3], etc.

More advanced communication systems such as LENS [4] and RUBRIC [5], provide a greater deal of mail management. The approach of the both mentioned systems is based on production rules to support automated actions. Such a rules provide real time mail filtering and routing, as well as finding and prioritizing of useful messages.

Following the latest achievements in electronic mail domain, we have been developing the knowledge based electronic mail system (KBEMS), with contribution on more functional operations, based not only on document characteristics, but on users description also.

The KBEMS is a comprehensive intelligent system, which power is based on the simple idea of using well-described users and documents. The strong user's and document's description (knowledge base) with help of advanced information retrieval and knowledge based management system, provides a wide set of functional mail operations, such as collecting the useful information about the documents in KBEMS, sending the documents without knowing the user's address and automatic actions including prioritizing, selecting, sorting, discharging and forwarding the documents.

In the following text, in the chapter 2. we would say more about mail objects: document and user, in the chapter 3. we expose general KBEMS features, chapter 4. is related to mail operations and in the chapter 5. we present the KBEMS software architecture.

2. MAIL OBJECTS

The most common mail objects, in general are messages [4,5,6]. Since we want to support more advanced mail system, including also a different types of mail objects, such as book, report, letter, etc..., we accepted more general concept: *a document*. Thus, we consider a message as simplified type of document. Following further system requirements, we also must take into account another important object: *a user*. Modeling the both objects according to their semantic components to be easily available, was the first task we were concerned with.

The both objects are determined by corresponding descriptions, consisting of the set of (attribute, value) pairs and by the rules describing user's and document's features.

2 1.Attributes

The attributes are considered as carriers of characteristics of documents and users. The most attributes have structured value types, but there are also attributes with unstructured values. This causes the attribute description of mail object considered as a *semistructural*.

For example, the value type of a document attributes *AUTHOR* or *DATE* is structured, while the value type of the attribute expressing a *CONTENTS* of document is unstructured, Fig 1.

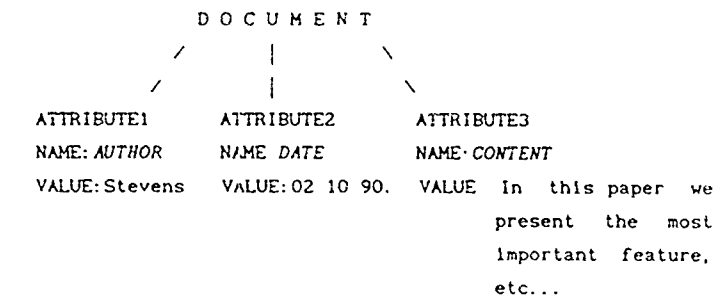


Figure 1.

Examples of the attributes presenting documents are
- with unstructured value type document content,
- with structured value types document identifier, author, date of creation, key words as a structured description of the unstructured document content, etc.

Examples of the attributes presenting users are
- with unstructured value type user biography,
- with structured value types user identifier, area of interest, profession, professional tasks, key words as a structured description of his biography

2.2.Rules

The other part of mail objects description is related to rules. The rules consist of a test and an action, if a document satisfies the test. the action specified by rule is performed Every user has the possibility to create his own rules, specifying the document's or user's attribute conditions and corresponding actions By gradually adding new rules, users can continually increase the helpfulness of the KBEMS.

The rules are globally classified in following groups

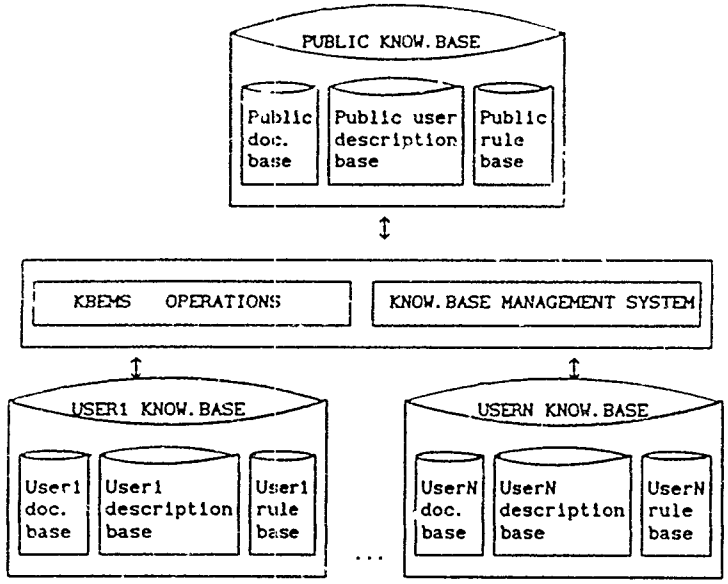
- the rules related only to documents, example.
  IF (the document is older than 10 days), THEN (remove it).
- the rules related to the both document and user, example·
  IF (the document is about operating systems), THEN (send it to the all users dealing with system programming).

The KBEMS rule are created using DIPSY-E language and expert system development tool [1,6,7].

3 CENERAL KBEMS FEATURES

In this Chapter we shall concentrate on the knowledge base cooperation aspects in KBEMS, as well as on the KBEMS features referring to final users

The mail objects are stored in the knowledge base, with possibility of gradually expanding it with the new attributes and rules. Every user has his own knowledge base consisting of user description, rule base and document base In fact, we are talking about *distributed knowledge base* system, which global architecture is shown on Fig 2



KNOWLEDGE BASED ELECTRONIC MAIL SYSTEM

Figure 2.

The knowledge base cocperation concept is based on the simple idea that electronic mail operation effects might be relevant to operation effected users, in a way that some rule conditions in their knowledge bases are satisfied. In fact, any operation causing relevant changes (more about that we will say in the Chapter 4.2.) activates the knowledge base management system [1,6,7], which, according to satisfied rule conditions, calls appropriate actions.

So. any user performing some operation could cause important changes in the knowledge base of any other user. This changes, controlled by knowledge base management system, are reflected in satisfied rule activation of the relevant users.

For example:

- USER1 *sent* a document to the all users interested in *electronic mail* (more about KBEMS operation we will say in the Chapter 3.),
- USER1, USER2 and USER3 received that document,
- under control of the knowledge base management system the satisfied rule actions of USER2, USER3 and USER4 are called , for example:
  - USER2 specified the rule for deleting documents from USER1,
  - USER3 specified the rule for sending documents from USER1 to USER5,
  - USER4 specified the rule for sending a massage to USER1, after receiving a document from him.

For currently implemented system we have designed the knowledge base management system to be activated only by operations called by user. Rule specified operations are performed automatically and they cause no knowledge base management system call. The reason for that is to avoid the explosion of knowledge base management system calls.

As also shown on Fig 2. there is a *public user*, who gathers the documents which might be interested to the greater number of real users (magazines, professional literatures and similar). Every user is allowed to read the public document and to send to the public user. The public user is presented to the KBEMS by his knowledge base, enabling thus automated rule based action to be performed under public documents. For example, all the documents arriving to the public users can be automatically forwarded to the all users, whose descriptions match the public documents's key words.

With exception of public documents, all the real user's documents are completely in property of their owners. In fact, the documents belonging to the certain user are quite invisible for the other users, but completely known to the system.

4. OPERATIONS IN KBEMS

The electronic mail operations supported by KBEMS, are globally classified in following groups:

- electronic mail objects creating,
- standard distribution operation (sending/receiving),
- selective distribution operations (selective sending/collecting),
- automated rule-based actions.

Creating of new documents and users is referred to:

- filling in the values of document attributes performed trough KBEMS - user dialog or on standard way by naming the document and writing its content, in which case the attributes are extracted by KBEMS itself,
- filling in the values of user attributes. performed trough KBEMS - user dialog, under control of KBEMS administrator,
- updating of the user attribute values: every user is allowed to change the content of his attributes,
- creating of rules. every user is allowed to create his own rules using DIPSY-E language and expert system development tool [1,6,7]

The operation send and the receiving operation set (delete, edit, sort, etc) are elementary mail services, common for every mail system. Hence, we will not detaily discuss them, although we developed them to complete our system. Let us only say that those KBEMS operation are a bit different comparing with the same operations supported by other mail systems The difference is related to adjustment for being called by knowledge base management system, providing the mentioned operations being activated automatically.

4 1.Selective distribution operations

The concept of selective distribution operations is based on semistructural description of documents and users, presented by attributes, as already mentioned in chapter 2. Due to representing the objects in such a way, we can easily found and use the relevant information they content.

For example, we can send a document reaching the users according to their attribute description, without specifying a certain addresses. We can also get the certain information about the relevant documents in KBEMS we do not possess.

Those operations, based on attribute mail object features, are called *selective distribution operations*. Before we say more about each of them, we shall prompt their common retrieval characteristics

The important feature supporting selective operations is a *matching alghorithm* The matching level for present implementation is reduced to one key word. Nevertheless, by algorithm modification, it

can be easily changed (30%, 50%, etc).

Since the matching is the important action concerning selective operations, it is more effective by using improved information retrieval method related to *thesaurus*. Thesaurus version in KBEMS consists of a synonym dictionary, synonym description base and appropriate procedure giving for the certain input an output presenting by the list of synonyms and synonym description. Fig 3 Thus, selective distribution operation for every key word invokes thesaurus to do retrieval more precisely.

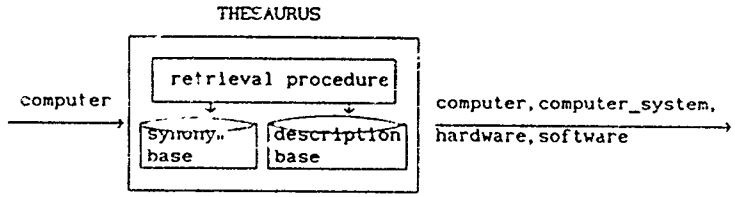


Figure 3.

4.1.1.*Selective sending*

Selective sending is modificated standard sending operation, in a way that receiver's address is not explicitly known. The potential receivers are specified by sending parameters, which actually present the user attribute description. All the users described in the knowledge base, whose attribute values according to the matching level, satisfy selective sending parameters, are real receivers.

The syntax of selective sending is:

SSEND name, parameter1 [,...,parameterN]

where *name* is a real name of the document being selective sent, while *parameters* are referred to the user attributes, such as area of interest, profession, professional tasks, key words of users receiving the document.

Examples of selective sending are as follows:

- Document XY send to the all users, who are dealing with COMPUTER SCIENCE and interesting in COMPUTER GRAPHICS, on SUN work station
- Document XY send to all RESEARCH ASSOCIATE, working on project DCIX

Selective sending calls in a currently implemented system for above examples are:

SSEND	SSEND
- *document name*: XY	- *document name*: XY
- *area of interest*: COMPUTER_SCIENCE	- *area of interest*:
- *profession*:	- *profession*: RESEARCH_ASSOCIATE
- *professional tasks*:	- *proffesional tasks*: DCIX
- *key words*: COMPUTER_GRAPHICS	- *key words*:

4.1.2. *Collecting*

Collecting is the opposite operation comparing to selective sending. In fact, this operation enables user to get information about the interesting topics referring to the all existing documents in the KBMES. This is done by specifying the topics, which after passing trough thesaurus, are matched against key words of all documents existing in the knowledge base. The result of this operation is presented by names and owners of documents, containing the key words, which, according to matching level, satisfies the retrieval.

The syntax for collecting operation is:

COLLECT parameter1 [,...,parameterN]

where the *parameter1,...,parameterN* represent the interested topics.

Example of collecting operation:

Collect all the documents with the key words DISTRIBUTION, COMMUNICATION.

The operation call in a currently implemented system for given example is:

COLLECT distribution, communication.

The result might appear as follows:

DOCUMENT NAME	DOCUMENT OWNER
D1	XXX
D2	YYY
D3	ZZZ

Only the documents collected from the public mail box are completely available to the user. On the contrary, the documents from the other mail boxes are protected, so only owners have acces, to them. To make this documents accessible to the certain user, the owner must be consulted.

4.2. Automated rule supported actions

The environment of electronic mail system described in this paper, allows users to build rules [3,4] for specifying requirements related to their mail.

The rules, already mentioned in chapter 2., are activated automatically, under control of knowledge base management system. The rule activation is initiated by certain KBEMS operations, which cause such a changes in the knowledge base, which might be relevant to the certain users. Actually, when such a changes happened, we say that an *event* is happened.

Example of the operation causes an event is sending. When the user sent a document, it might be interested not only for the receiver, but for some other users also. So, sending the document must be announced to the knowledge base management system, in order to call the actions specified by satisfied rules. Another example of event causing operation is selective sending.

There are also operations which cause no event. For example, if a user deletes his document, this operation is not relevant for the other users. Hence, this would cause no event and no call of knowledge base management system. Another examples of such operations are: editing, sorting, collecting, etc...

In fact, always when an event happened, the knowledge base management system is called. This causes retrieval of a rule base of every users in the KBEMS, to found out which actions are to be taken.

The knowledge base management system is a part of expert system shell DIPSY-E. It is detailly described in [1,6,7,].

The actions specified by rules are sending, selective sending, prioritizing, sorting and deleting the documents (the last three are contained in receiving set of operations). Some rule examples are as follows:

- IF (document is *older* than 10 *days*) THEN (*remove* it),
- IF (*sender* of document is the *chief*) THEN (*prioritize* it),
- IF (received document has the *key words image, pixel*) THEN (*send* it to all users whose *area of interest* is *graphics*),
- IF (the user is *upsent*) THEN (*send* the information about that to the all senders addressing mail to his mail box).
- IF (received documents has the *key words office_automation*), THEN (*send* it to all users engaged with *project DCIX*).

5. GLOBAL SOFTWARE KBEMS ARCHITECTURE

The global software system architecture is shown on Fig 4

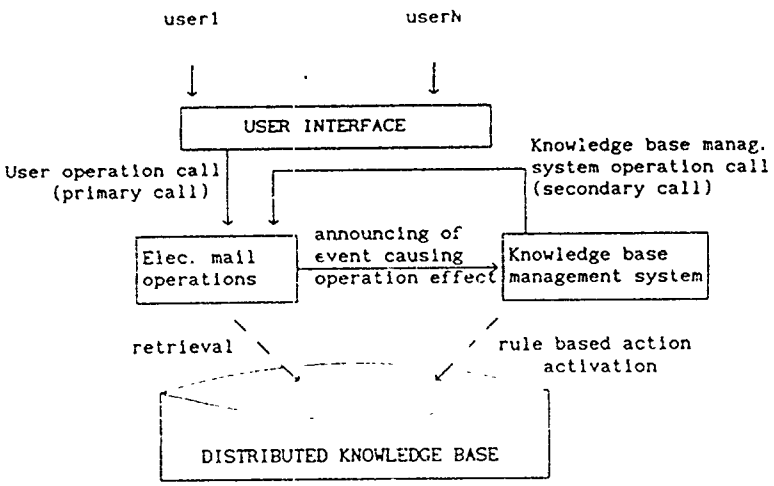


Figure 4.

The KBEMS has being developed on the multi-user computer system under UNIX operating system.

Let us now illustrate the steps of selective distribution operation given by the first example of selective sending, mentioned in the chapter 4.1 1 .

1. the sender initiates the operation (primary call),

2. after getting key words synonyms (if any), all the users knowledge bases (refered to attribute descriptions) are retrieved in order to find those, whose attribute values, according the matching level, satisfied the attribute values required by the operation parameters,
3. the document has been sent to the found users,
4. the operation effect is announced to the knowledge base management system ,
5. the knowledge base management system is activated in order to find satisfied rules,

As a rule example we take the last one from the Chapter 4.2:

6. the knowledge base management system invokes selective sending operation (secondary call),
7. after the document is sent to all users engaged with project DCIX, the process has been finished,
8. the next whole process activation is started by the next event causing operation.

CONCLUSIONS

In this paper we presented the comprehensive electronic mail system providing selective distribution operations and automatic rule-based actions.

We have seen in this paper how a combination of ideas from artificial intelligence and information retrieval can provide the powerful computer-based communication and coordination system We have also seen the importance of not only a document being mailed, but also the importance of a user model, which helps us improving system capabilities in a great deal.

LITERATURE

1. A. Filipovic, M Kantardzic, H. Glavic, N. Gujic, V. Okanovic, "DIPSY-E Expert system developing tool", I Yugoslav Conference of Artificial Intelligence Accomplishment and Application, Dubrovnik, Octoba , 1989.

2. Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, W. Shaaf, Raymond S. Tomlinson, Virginia M. Travers DIAMOND: A MultimediaMessage System Built on a Distributed Architecture, IEEE 1985.

3. Najah NAFFAH, Director of the KAYAK project: Distributed Office Based Document Generatior., Online Conference - "Local Networks and Discributed Utilization, Elsevier Science Publishers B.V IFIP, 1985.

4. Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst, Michael D. Cohen: Intelligent Information-Sharing Systems, Communications of the ACM, VOL. 30, No 5, May 1987.

5. Erian P. Mc Cune, Richard M. Tong, Jeffrey S. Dean, Danilel G. Shapiro: RUBRIC: A System for Rule-Based Information Retrieval, IEEE Transactions on Software Engineering, Vol. SE-11, No 9, September 1985.

6. Buchanan, Shortliffe. "Rule-Based Expert Systems", Addison-Wesley, '84.

7. M. Kantardzic, N. Gujic, A. Filipovic, V. Okanovic: Tightly-coupled rule-search strategy, Computer on the University, Cavtat, Juny 1990.

8. Thomas W. Malone, Kenneth R. Grant, Kum-Yew Lai, Ramana Rao, David Rosenblitt: Semistructered Messages are Surprisingly Useful for Computer-Supported Coordination, ACM Transaction on Office Information Systems, Vol. 5, No 2, April 1987.

9. Harold E. O'Kelley: Electronic Message System as a Function in yhe Integrated Electronic Office, AFIPS Cnoference proceedings, May 19-22, 1980, Anaheim, California.